BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

NETWORK SERVICE DIAGNOSTICS BASED ON PACKET ANALYSIS

DIAGNOSTIKA SÍŤOVÝCH SLUŽEB ZALOŽENÁ NA ANALÝZE PAKETŮ

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR **Ing. MARTIN HOLKOVIČ**
AUTOR PRÁCE

SUPERVISOR **doc. Ing. ONDŘEJ RYŠAVÝ, Ph.D.**
ŠKOLITEL

BRNO 2023

Abstract

This dissertation is a compilation of peer-reviewed papers published during the doctoral study. The papers, as well as the thesis itself, focus on the problem of fault diagnosis in computer networks. Network faults can significantly affect the performance, availability, and security of networks, resulting in significant financial losses and degradation of user experience. To mitigate these problems, network diagnostics has become a critical area that focuses on identifying, locating, and resolving network faults. Although computer network diagnostics is not a new field and there are several tools and publications available, it is still an unsolved problem.

The goal of this work was the development of new diagnostic methods, each of which is focused on a different type of error, and in the whole the individual methods are complementary to each other. The developed methods aim at automating the diagnostic process, which is time-consuming in case of manual execution and requires good knowledge of network protocols and technologies. Network packets saved in PCAP format are used as a data source for diagnostics, which allows to detect errors without the need to interact with the network. Another advantage of the developed methods is their wide applicability, as the solution is not limited to a small number of network protocols, and at the same time the methods can be used by network administrators and not only by researchers or programmers.

An integral part of the thesis is a description of the contribution of the dissertation and the results achieved during the study. In addition to the acceptance of the published articles, the successful commercialization of the product Flowmon Packet Investigator proves the correctness of the chosen approach. Finally, the research areas are summarized and further steps to improve the integration of the results into practice are presented.

Keywords

computer networks diagnostics, troubleshooting network errors, diagnostic methods

Reference

HOLKOVIČ, Martin. *Network Service Diagnostics based on Packet Analysis.* Brno, 2023. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Ondřej Ryšavý, Ph.D.

Rozšířený abstrakt

V dnešním propojeném světě hrají počítačové sítě klíčovou roli při usnadňování bezproblémové komunikace a výměny informací. Chyby a narušení sítě však mohou významně ovlivnit výkon, dostupnost a bezpečnost těchto sítí, což vede ke značným finančním ztrátám a zhoršení uživatelských zkušeností. Pro zmírnění těchto problémů se diagnostika sítě stala kritickou oblastí, která se zaměřuje na identifikaci, lokalizaci a řešení síťových chyb. Tato práce zkoumá základní aspekty síťové diagnostiky a zdůrazňuje její význam pro zajištění spolehlivých síťových operací. Začíná stručným popisem samotných sítí, možností jejich monitorování a způsoby zpracování síťových paketů. Následně práce popisuje důležitost diagnostiky chyb, popisuje různé typy síťových chyb a možnosti automatických nástrojů pro diagnostiku.

Hlavní kapitolou této práce je shrnutí výzkumu, která obsahuje šest rozličných diagnostických metod, které byli publikovány v osmi článcích. Všechny tyto články a metody se zaměřují na pasivní diagnostiku ze zachycených síťových dat ve formátu PCAP. Pro zpracování dat v tomto formátu je využit nástroj TShark, který je verzí aplikace WireShark pro příkazový řádek. Výhodou využití nástroje TShark je obrovská podpora síťových protokolů. Při tvorbě metod byl kladen důraz na přesnost a vysvětlitenost výsledků. Metody pro nasazení nevyžadují žádné, nebo jen velmi malé množství, trénovacích dat. Dále metody umožňují rozšiřovat znalostní bázi, kterou je následně možné aplikovat na libovolný protokol podporovaný nástrojem TShark.

První diagnostická metoda pojednává o použití algoritmu rozhodovacího stromu pro diagnostiku síťových problémů. Přístup rozhodovacího stromu umožňuje simulaci akcí skutečných správců sítě pomocí snadno srozumitelných pravidel. Metoda je však omezena na předem definované chyby a může vést k nepřesným závěrům pro neznámé chyby. Rozhodovací strom je rozdělen na vnitřní uzly, které vyhledávají konkrétní události v datech, listové uzly, které identifikují příčinu problémů, a systém, který spojuje jednotlivé uzly pro umožnění průchodu stromem. Každý uzel stromu je spojen s pravidlem a nové problémy lze pokrýt vytvořením dalších pravidel. Rozhodovací strom je rozdělen na podstromy, z nichž každý představuje jeden protokol. Doba zpracování u větších vstupů je problémem a jsou diskutovány optimalizační techniky. Použití indexace vstupních dat, kromě zvýšení výkonu, umožňuje integraci s analýzou logovacích souborů. Navzdory vzestupu přístupů strojového učení prokázal přístup založený na pravidlech svou použitelnost a byl integrován do komerčního produktu Flowmon Packet Investigator.

Druhá metoda popisuje aplikaci bezpečnostní analýzy při diagnostice chyb souvisejících se síťovými útoky. Vytvořená metoda dokáže detekovat útoky napříč různými toky bez omezení na vybrané pakety nebo protokoly. Detekční algoritmus rozděluje pakety do skupin na základě atributů paketů a následně aplikuje operace a podmínky pro hledání specifických atributů. Když jsou podmínky v rámci skupiny splněny, počítadlo se zvýší, a pokud dosáhne předem definované hodnoty, je detekována požadovaná událost. V porovnání s klasickými IDS řešeními umožňuje metoda flexibilní vyhledávání událostí napříč více toky, ale za cenu pomalejší rychlosti zpracování. Výsledek je tak určen spíše k doplnění, než nahrazení stávajících řešení IDS. Tato metoda také nabízí uživatelsky přívětivý jazyk pravidel pro administrátory pro rozšíření seznamu událostí, které mají být detekovány. Pro zvýšení využitelnosti pro diagnostické účely je součástí výstupu nástroje seznam paketů přispívajících k detekované události. Funkčnost metody pro diagnostické účely byla otestována vytvořením 35 pravidel různé složitosti.

Třetí metoda je založena na pravidlech pro detekci a identifikaci síťových problémů pomocí analýzy vizuálních reprezentací síťových dat. Metoda se zaměřuje na identifikaci vzorů

v datech, jako jsou špičky nebo křížení linek v grafu, které indikují anomálie nebo chyby v síťové komunikaci. Tento přístup automatizuje manuální analýzu prováděnou správci sítě pomocí vizuálních reprezentací síťových aktivit. Využívá jednoduché popisy změn hodnot a nespoléhá se na algoritmy zpracování obrazu. Proces zahrnuje přiřazení paketů k časovým intervalům, použití agregačních funkcí a vytvoření řetězcových reprezentací dat grafu. Vzory jsou pak identifikovány vyhodnocením regulárních výrazů nad těmito řetězci. Funkčnost tohoto nástroje je demonstrována na příkladu výstupu, který ukazuje detekci vzoru skokových skoků souvisejících s QoS řazením do fronty a nastavením ukládání do vyrovnávací paměti, které může ovlivnit uživatelskou zkušenost v multimediálních aplikacích.

Čtvrtá metoda je zaměřena na využití předchozích diagnostických výsledků s cílem odhalit a diagnostikovat opakující se chyby. Tento přístup zahrnuje vytvoření specifického modelu pro každý aplikační protokol, zachycení správného chování protokolu pomocí dvojic zpráv žádost-odpověď. Model je rozšířen o chybné přechody a popisy chyb poskytnuté administrátorem. Během diagnostiky model identifikuje neznámé chyby, když narazí na stav bez odpovídajícího dalšího přechodu. Součástí metody je také použití modelů časovaných automatů, které u přechodů navíc zahrnují časové informace k pokrytí více situací a problémů. Metoda byla testována na různých protokolech s dosahováním přesnosti od 63 % do 100 %. Zatímco model může vyžadovat neustálé učení, aby pokryl všechny možné situace, pomáhá diagnostikovat opakující se chyby a šetří čas zkušeným správcům. Naučené modely lze sdílet, což umožňuje méně zkušeným správcům těžit z odborných znalostí ostatních.

Cílem páté metody je identifikovat typografické chyby v uživatelsky zada-ných datech, jako jsou například názvy serverů, čísla portů nebo přihlašovací jména. K odhadu zamýšleného slova využívá metoda heuristiku založenou na minimální vzdálenosti úprav a běžných typech typografických chyb. Metoda detekuje chybně napsaná slova, vygeneruje sadu možných správných slov a vyhodnotí každého kandidáta pomocí vytvořené heuristiky. Slovo s nejvyšším skóre určujícím pravděpodobnost je vybráno a nahlášeno administrátorovi. Metoda se opírá o slovník správných slov a nebere v úvahu gramatiku jazyka ani kontext okolního textu. Vyhodnocení ukázalo, že tento přístup je účinný pro data aplikační vrstvy, ale méně vhodný pro data nižší vrstvy, jako jsou IP adresy a transportní porty.

Šestá metoda se zaměřuje na omezeních současných nástrojů pro analýzu sítě a navrhuje nový přístup pro analýzu na agregované úrovni dat. Stávající nástroje postrádají dostatečné informační a vizualizační možnosti. Navrhovaný přístup využívá metaforu filtračního trychtýře, kde je aplikováno více vrstev filtrů, aby se snížilo množství dat pro analýzu. Filtrovací filtrační vrstvy jsou navzájem zcela nezávislé a je možné je aplikovat na libovolné síťové data, jako například IP adresy, čísla protokolů, nebo čísla portů. Nástroj podporuje úlohy, jako je identifikace zdrojů komunikace, detekce neobvyklých vzorců, identifikace uzlů a služeb a sdílení parametrů analýzy. Tento nástroj byl navržen pro podporu manuální vizuální analýzu ze strany správců sítě. Byl vyhodnocen zapojením skutečných správců sítě, kteří pomocí tohoto nástroje prováděli úkoly, a jejich zpětná vazba byla většinou pozitivní, zdůrazňující jeho flexibilitu a podporu během počáteční analýzy zachycování paketů.

Kromě recenzovaných článků byli některé nástroje vydány jako software. První metoda, založena na principu rozhodovacích stromů byla úspěšně komercializována společností Flowmon Networks do produktu Flowmon Packet Investigator, který je ve formě placených licencí poskytován zákazníkům uvedené společnosti. Kromě toho vznikl na základě metody zaměřenej na vizuální analýzu síťového provozu na agregované úrovni výzkumný projekt s označením SECURIAN (TAČR FW06010009), kterého cílem je kromě jiného integrovat poslední popisovanou metodu do analytického nástroje Flowmon Monitoring Center.

Network Service Diagnostics based on Packet Analysis

Declaration

I hereby declare that this PhD thesis was prepared as an original work by the author under the supervision of doc. Ing. Onřej Ryšavý, Ph.D. I have listed all the literary sources, publications, and other sources used during this thesis's preparation.

.....
Martin Holkovič
July 17, 2023

Acknowledgements

At first, I wanted to skip this part, but then I realized that I actually deeply thank my supervisor and respected leader Ondra for the effort and trust he put in me and for his time that he could have used for better purposes.

Contents

Chapter 1

Introduction

Computer networks are complex systems consisting of a large number and variety of devices and applications that communicate together by using different communication protocols. The complexity of networks rapidly rises by introducing new types of technologies, such as Software-defined networks, the Internet of Things, or computer network virtualization. In such a complicated environment, it is just a matter of time before an error occurs that can negatively affect the network's functionality [62]. For example, errors can be caused by a hardware failure, a misconfiguration, or a software bug [54].

Due to an error, a network service may become unavailable, network performance may degrade, the user experience may decrease, or an error can cause a financial loss in the case of business networks [122]. In huge networks like YouTube, CloudFlare, or Paypal, the downtime is worth hundreds of thousands of dollars per hour [38]. Even after users report a problem inside the network and administrators start working on it, it takes some time to solve it. According to an administrator survey, over 50% of administrators say that the average time required to solve a single problem is more than 30 minutes [139].

The process of searching for network errors, finding causes of errors, and fixing errors are called network troubleshooting or diagnostics. Usually, end users need help to solve problems. It is the role of network administrators to diagnose errors, and administrators spend a significant amount of time on it. Because computer networks are complex systems, diagnostics require in-depth knowledge of individual network parts and technologies. Even if an administrator has enough knowledge of all the necessary technologies, it is time-consuming and requires much effort [106].

Existing tools usually only detect the problem's symptoms, and it is up to the administrator to understand and fix the problems. Another complication is that several data sources need to be checked to find a real source of the problem. It amounts to log files, NetFlow records, or captured packets. According to a short survey [106], network administrators would like to have an automated monitoring tool to diagnose network errors that would be able to identify such problems. Previous researchers have developed methods that help network administrators to diagnose faults and performance problems. However, they require installing agents to hosts, providing rich information about the environment, or an active monitoring approach [69].

Even with the help of the current network monitoring tools and published techniques, network diagnostics is a primarily manual activity [107]. A common way of manual diagnostics is using a network packet analyzer (e.g., Wireshark). The analyzer processes captured network traffic and decode individual packets. The administrator analyzes available information and compares the data with expected values (e.g., based on RFC standards). This

manual process is time-consuming and requires a good knowledge of network protocols and technologies.

This work's motivation is based on a research project between the CESNET organization and the Flowmon Networks company, which provides a solution for monitoring computer networks. The project aimed to create an automated diagnostic tool that Flowmon Networks could deploy to its customers. The company's primary customers are companies with about 100-1000 users. The monitoring is based on special probes that process the copy of the network traffic. An example of a monitored network is shown in Figure 1.1.

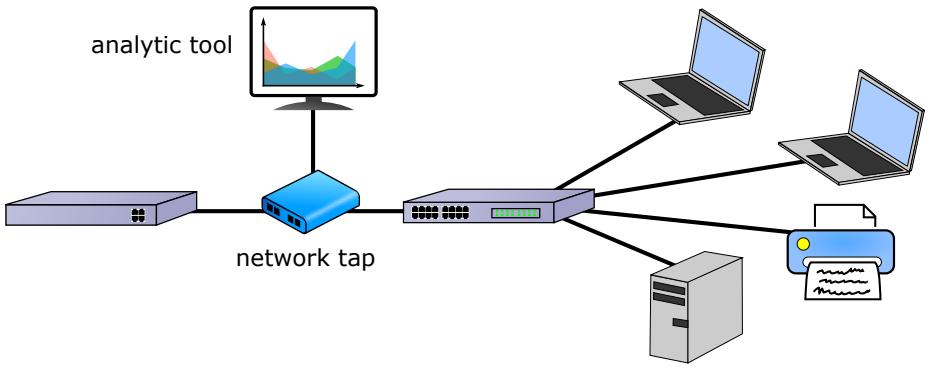

Figure 1.1: The Network monitoring architecture in which the traffic is copied from the network into the monitoring station.

Using monitoring probes deployed inside a network gives a network administrator full access to the transferred data. By correctly placing the probes inside the network, an administrator can monitor any location inside the network, e.g., web server or end-user network segment [70]. Network traffic is usually not analyzed inside those probes directly, but the traffic is sent to a central server. The administrator can use this access to capture and analyze network traffic in the case of a network problem. By analyzing the traffic captured in this way, network administrators can diagnose network problems and ensure the functionality of a network [30].

The goal and the contribution of this work are to design and validate diagnostic methods capable of root cause identification of network-related errors. These methods would require only captured network traffic and not send any additional traffic into the monitored networks. The methods are focused on the following:

- simulating the steps of a real administrator,
- the way how to detect complex security events,
- simulating pattern search in chart data,
- how to reuse the previous diagnostic results,
- how to detect typing errors in end-user data,
- improving the flexibility of visual top-level analysis.

1.1 Research Goal and Objectives

This thesis aims to advance the network troubleshooting field by designing and experimentally developing an automated computer network diagnostic system. The system should cover major Internet protocols and be able to detect different types of errors by providing only a tiny amount (hundreds to thousands of packets) of captured network communication. The passive detection approach is considered not to interfere with regular network operations. The following research objectives can refine the goal:

1. **Accurate.** Results of all created methods must be correct and accurate. Only in that situation, the methods can be trusted by network administrators and deployed in their networks.

2. **Extensible.** The system can be updated to extend the set of detectable errors using a simple (declarative) language. Using the domain-specific language enables users to define new rules of the system, thus enabling them to extend the system with errors specific to various environments.

3. **Protocol support.** Each well-known and commonly used protocol must be supported regardless of the network topology. There should also be no difference between binary or text protocols.

4. **No need for a huge dataset.** The created methods can only expect a small amount of network traffic for analysis. It should be expected that the capture filter was applied to record the traffic. This may lead to incomplete information.

5. **Passive method.** No traffic should be sent to the monitored network during the diagnostic process. The methods need to work only with the provided data.

1.2 Assumptions and Requirements

This thesis focuses on new diagnostic methods for detecting errors by analyzing computer network communication. These new methods aim to provide valuable information to network operators and administrators by identifying the error and providing the probable cause of failure. Therefore the amount of effort and time required to correctly find the root cause of a problem can be reduced. There are several assumptions, requirements, and practical considerations that should be taken into consideration for the system design. Designing a general diagnostic method for computer communication requires us to assume different technologies, environments, devices, and applications. To make this problem tractable, this section specifies assumptions made and explicitly defines requirements to be met by the target methods.

1.2.1 Environment

The Internet consists of many interconnected networks built on top of the TCP/IP protocol suite. However, the technology that they are made of may vary. Several research works aim to detect errors in these network technologies, which can often be solved by providing a specialized tool that implements domain knowledge to identify the anomaly situation and explain it. The aim of the present work is different. The focus of the proposed diagnostic methods is on Internet communication in enterprise or SOHO networks. These networks

provide connectivity for end users and Internet services. This requires analyzing the entire TCP/IP stack and seeking non-standard or erroneous data exchanges to identify possible malfunctioning and errors.

1.2.2 Network Protocols

The main source of information for the proposed diagnostic methods is the information extracted from network packets. Network protocols control network communication. One needs to understand the network protocol to extract relevant information from the communication. It is necessary to choose such a way of processing network data that will support a huge number of protocols. Two main approaches exist: create a custom solution for processing data or use an existing solution. Both approaches have advantages and disadvantages. For example, tools highly oriented to a processing speed instead of flexibility and usability usually implement their own network data parsers. The reason is that they can optimize the whole processing unit according to future data processing. On the other hand, an approach that uses existing processing tools can implement only a few protocol parsers.

1.2.3 Traffic Analysis

The diagnostic methods primarily use PCAP files that contain captured packets as the data source. Processing of the captured traffic has the benefit that the processing can not affect the monitored (production) network, and the analysis can be executed in any location at any time. The created methodology must work regardless of the network topology, technologies, and protocols inside the network. Most notably, it is necessary to process data even if they are tunneled or segmented correctly.

To use the created methods in real-world scenarios, the methods must only require a little traffic to find the error's root cause properly. This limitation arises from several reasons: privacy - because one user has a problem, it is not possible to capture traffic of all network users; practicality - the simplest way of capturing traffic is to place a capture probe on the edge of the network (end users or servers segment) and capturing traffic from multiple places would require multiple capturing probes; performance - by capturing a specific subset of traffic (e.g., communication between one user and one server), amount of processed traffic is significantly decreased that allows using complex algorithms.

1.2.4 Automation

The proposed methods' primary purpose is to speed up the diagnostic process by allowing automated execution. With algorithmic diagnostics, each captured network traffic can be processed by multiple parallel methods, so the administrator does not need to figure out which method should be executed to diagnose the currently analyzed problem.

1.2.5 Explainability

It is essential to create new diagnostics methods that real administrators would use, not just the researchers. Creating methods must be easily extendable even without programming skills to achieve this. For example, an administrator would instead work with a rule-based system that uses a simple language instead of reimplementing neural networks created by a machine learning algorithm.

The created methods need to produce results that are explainable to an administrator and can be traced back to know why the system produced such results. The methods must be deterministic, and if a diagnostic result is provided, the administrator needs to know how and why the system deduced such a result. Another benefit of the deterministic approach is that the system ends with the same input data with the same results. If multiple administrators are working on the same problem in parallel, they can collaborate.

1.3 Thesis Outline

The presented thesis has form of a collection of seven peer-reviewed papers written by the author as the major contributor. The thesis is organized as follows. Chapter 2 provides a summary of state of the art and presents the necessary background information in the area of network monitoring, network troubleshooting and error diagnostics. Chapter 3 summarizes the research contribution that consist of six different methods, and lists the outcomes made during the doctoral study. Chapter 4 concludes the thesis by discussing the contributions, comparing the proposed methods, and listing the outcomes.

Chapter 2

State of the Art

This chapter describes the state of the art that covers the basics of computer networks and how they are monitored and diagnosed. The chapter starts by describing computer networks and how they are monitored. From the diagnostic perspective, it is essential to understand methods used for monitoring as it lately affects the diagnostics process. The chapter focuses on analyzing the network packets as all the new proposed detection methods described in the next chapter use network packets as data sources. The last part of the chapter describes the diagnostic process and talks about the various implementations of the current solutions.

The state of the art has been created based on research papers that were found from the IEEE Xplore, ACM Digital Library, and Google Scholar libraries. Only papers that were found by searching the keyword "network diagnostics" or "network troubleshooting" and that were published after the 2010 (including) have been analyzed. In addition to all these papers, in some cases a cross reference lookup has been used - a paper cited by the analyzed paper or a paper that cited the analyzed paper.

2.1 Computer Networks

A computer network is a set of interconnected devices that can communicate between themselves. In addition to most typical devices like personal computers or servers, computer networks may connect printers, phones, network devices, industrial devices, cameras, light bulbs, and many others. The primary purpose of computer networks is to allow all these heterogeneous devices to communicate with each other, no matter the distance or the technology being used. Since the emergence of the WWW protocol in the 1990s [13], which has massively spread computer networks among ordinary people, much time has passed. Nowadays, computer networks are inseparable parts of our society, and many aspects of our daily lives depend on the functionality of those networks. For example, computer networks are necessary for electronic payments, online meetings, cloud computing, and social media. Figure 2.1 shows the number of IoT devices connected to computer networks in billions according to Statista[1]. If we use the number of devices in the year 2023 and 8 billion people as the assumed number of people on Earth in the year 2023, we get that, on average, there are over six devices (just IoT) for one person.

[1]S. R. Department, "Internet of Things - number of connected devices worldwide 2015-2025," 2016, (Accessed: 2021, Feb 2). [Online]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

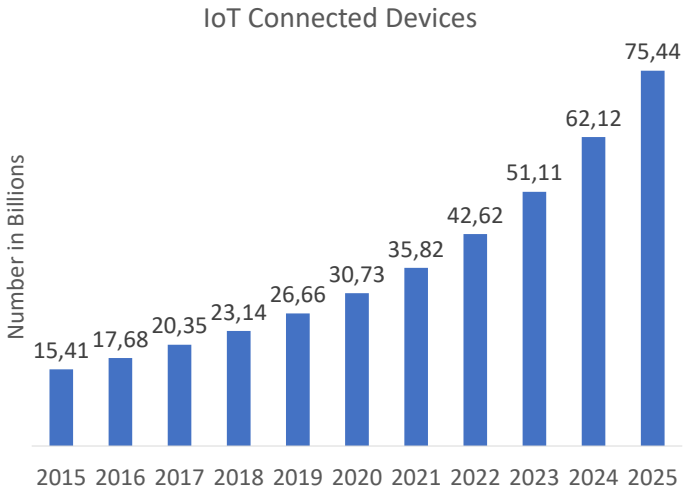

Figure 2.1: The number of IoT devices according to the survey conducted by Statista.

Adding new and new technologies and use cases to the networks caused the complexity of networks rapidly increase [88]. At the same time, other requirements have been introduced. More than being able to communicate with other devices is required nowadays. Users expect low latency and high speeds of data transfers, safety and privacy for the users, and no outages and high availably. The combination of higher requirements with the higher complexity of the networks leads to higher requirements placed on network administrators, which ensure that the networks are working as expected.

There are many ways to categorize networks into categories. The most common classification is based on the size of the network: Local Area Network (LAN) is for networks within a limited area, such as a building or a campus, and Wide Area Network (WAN) that connects multiple LANs into a large network that spans a large geographic area such as country or even the world, and Metropolitan Area Network (MAN) that covers a geographic area larger than a LAN but smaller than a WAN, such as a city. Another classification can be based on used link technologies: wired like metallic, optic, or wireless like WLAN or 5G networks. Some networks can even be classified based on their usage, resp. the used protocols. For example, Industrial Control System (ICS) networks, Software-Defined Networks (SDN), or networks for the Internet of Things (IoT). This thesis is not specific to any of these categories, and therefore when talking about computer networks, the thesis talks about all of these.

2.1.1 TCP/IP Model

To provide a standardized approach to network communication that could work across different types of devices and technologies, a TCP/IP model has been created. The TCP/IP model is based on the concept of layering, where each layer is strictly defined and provides a different set of functions to the higher layer. This layered approach allows for the independent development of network protocols as well as better scalability in computer networks. The TCP/IP model consists of four layers:

1. **Application layer**: This is the highest layer and contains protocols used by applications that provide services to end-users and other applications. Because of the protocols on this layer, a device can understand data from other devices that use

the same protocol. For example, one of the most common protocols is the HTTP protocol, used by web browsers to allow users to display content on web servers.

2. **Transport layer**: Protocols on this layer are responsible for transporting data from one application to another. As each machine can use hundreds of applications that communicate over the network simultaneously, it is necessary to deliver data to proper applications. The most commonly used protocols are TCP and UDP. In the case of TCP, the protocol can ensure that all data sent from one application is delivered in the same form - no errors, gaps, no duplicates, and in the same order.

3. **Internet layer** (sometimes called Network layer): The layer is responsible for transporting data from one device to another. Precisely, this problem consists of addressing and routing between devices and networks. The most common protocol of this layer is the IP protocol.

4. **Network access layer** (sometimes called Physical layer): The lowest layer that is responsible for providing physical connectivity between two different devices. The most crucial problem the layer deals with is addressing devices within the same network and hardware standards that specify how to transmit data over various physical link technologies.

Each computer network layer has a distinct set of functions to perform. For the network to operate effectively, all four layers must work in conjunction with each other. If a single layer fails, it can adversely affect all the layers above it. Therefore, it is essential to ensure the proper functioning of each layer for optimal network performance.

2.2 Network Monitoring

Monitoring computer networks is an essential activity used by network administrators to see the current status of computer networks and ensure that networks work according to the defined policies. With network monitoring, administrators can detect and diagnose issues such as bottlenecks, packet loss, network congestion, security threats, and other anomalies affecting network performance. Additionally, network monitoring can help organizations to comply with regulatory requirements related to data privacy and security by detecting and reporting any unauthorized access or activity on the network. Overall, computer network monitoring is a crucial aspect of network management and requires a combination of techniques and tools to effectively monitor and maintain the performance and security of network infrastructures [111, 46, 32].

Network monitoring involves collecting and analyzing data from various sources to gain insights into a computer network. The sources can be classified into one of three different approaches [144, 68, 117]:

1. **Passive monitoring**: This approach contains techniques that are non-intrusive and do not interfere with the network devices and their network traffic. The most common data sources used in passive network monitoring are:

   (a) **Network devices**: When an administrator connects to the network device, such as a router, switch, or firewall, the administrator can get information directly from the devices [1, 132]. For example, the routing table from the router can be used to check how the traffic is being forwarded. In addition to connecting

a user to the device, retrieving data from devices remotely using a network protocol such as SNMP is possible. SNMP can retrieve information such as the number of transferred bytes for all network interfaces [4, 79].

(b) **Network traffic**: Another common data source is working with data being transferred inside the network [27, 103, 58, 2, 6, 77, 74, 19, 64, 36, 104]. In the basic form, it means capturing and analyzing packets by duplicating data from the physical links or inside the network devices. Because of the enormous amount of data, it is more common to work with aggregated data formats such as NetFlow/IPFIX records [45, 89, 90, 143].

(c) **System logs** [21, 51, 37, 55, 54, 84, 137, 83, 142, 96, 24, 145, 146, 79]: This data source is mainly used on the end stations (including the servers). Most applications record (or can be configured to record) all the activities the application is doing, including the interaction with computer networks. These records are called log files or just logs and are usually used by administrators when the application is not working as expected. It is also a common practice to send logs to a central location with a protocol called Syslog.

2. **Active monitoring**: Active network monitoring is an intrusive approach that involves sending test traffic across the network to the end stations to measure its performance and identify any issues [93, 73]. Examples of such data sources are:

   (a) **ICMP protocol** [105, 49, 3, 78]: ICMP is a network protocol used by network devices to communicate error messages about network conditions. The most common tool using this protocol is *ping* which checks if there is a connection between two devices.

   (b) **TCP protocol** [85, 102]: By analyzing the TCP protocol, metrics such as throughput, packet loss, and latency can be obtained.

   (c) **Application protocol** [43, 49]: By sending a predefined set of packets, it is possible to imitate a real user interaction with any application and measure the interaction's performance and result. For example, it is possible to simulate a user trying to browse the content of the HTTP server.

   (d) **Security scanners** [123, 66]: Similar to the previous data source, however, in this case, the goal is not to simulate just the user's application data but to check the security (or vulnerabilities) of the devices. By comparing which services should be available and which the security scanner can reach, it is possible to verify the correctness of the applied security rules.

3. **Hybrid monitoring** [48, 115, 39]: Hybrid network monitoring combines passive and active approaches to provide a more comprehensive view of network activity. This approach captures and analyzes passive data sources and sends test traffic inside the network. An example of such an approach can be to send test traffic only in case the passive monitoring detects an unusual situation inside the network.

Generally, it is impossible to tell which data source is the best and which is the worst. The choice depends on many factors, such as the network size, law regulations, the organization's needs, and the network type. For example, passively monitoring transferred data may be more appropriate for security monitoring, while active monitoring may be more appropriate for performance monitoring. Of course, in real situations, multiple approaches and

data sources are used simultaneously. With information from all these sources, network administrators can identify network issues, take steps to optimize the network's performance, enhance security, and improve the user experience.

When network monitoring involves capturing packets or logs for extended periods of time, it can generate a significant amount of data. To minimize the amount of data generated by network monitoring, various techniques can be utilized:

1. **Filtering**: One of the most effective ways to reduce the amount of data is to filter out data irrelevant to the specific use case. For example, to monitor a web server, it may be enough to monitor just the HTTP traffic to or from the specific server, and all other communications can be dropped.

2. **Sampling**: Sampling involves capturing only a subset of the packets. When monitoring a web server, monitoring only every nth connection may be enough.

3. **Truncation**: This involves capturing only the first few bytes of each packet or the first few packets of each connection, which can be enough to analyze what is happening. When monitoring a web server and a user starts downloading a file with the size of 1GB, it is not necessary to store all the data. Only the first few packets containing all the headers can be enough.

4. **Aggregation**: Aggregation involves combining multiple entries into a single entry. This can be useful for reducing the number of entries generated while still providing helpful information. In network monitoring, the most common aggregation is the NetFlow monitoring approach which aggregates all packets from a connection into one single entry. When monitoring a web server, each HTTP request would be represented by one data record instead of many network packets.

5. **Retention policy**: This policy defines how long to retain the captured data. As computer networks change constantly, the value of data used for monitoring the current state decreases as they age. For example, to analyze the current performance of a web server, it is irrelevant to have one year old data.

Using these techniques can reduce the amount of data generated by network monitoring while providing helpful information for network troubleshooting, security monitoring, or other purposes. However, balancing between reducing data and maintaining sufficient data to provide valuable insights is important.

Administrators can monitor networks at various points in the network, depending on the goals and requirements of the monitoring process. Figure 2.2 and the following lists include the most common locations:

1. **End stations**: This includes users' computers, servers, printers, or IoT devices.

2. **Core network**: The core of the network is the central part of a network that facilitates data exchange between multiple parts of the network.

3. **Datacenter**: Not all traffic must go through the network's core. Communications between servers usually stay inside the data center part of the network. As these communications are usually essential, monitoring this part of the network makes sense.

4. **Network perimeter**: It is the border where one network ends and another begins. A typical example is the connection to the Internet service provider.

5. **Cloud services**: Nowadays, the trend is to move from on-premise deployment of applications to cloud deployment when external companies host applications. Monitoring on this point makes it possible to monitor how users communicate with service providers and work with the hosted applications.

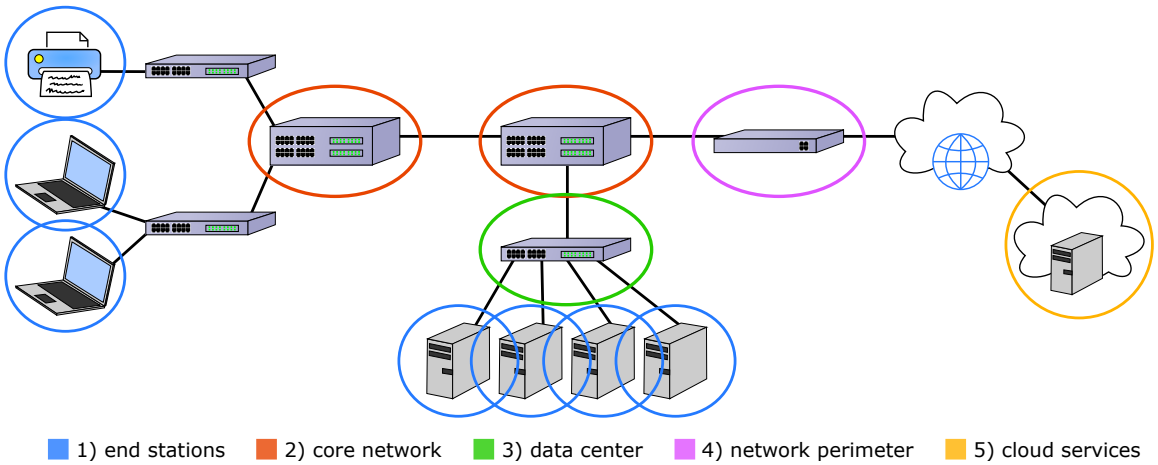

Figure 2.2: Visualization of various monitoring points in a network.

While network monitoring is an important part of network management and security, several challenges and potential problems are associated with it [68]:

1. **Data overload**: Monitoring tools can generate a large amount of data, which may overload network administrators and may even be impossible to analyze all of it.

2. **Inaccuracy**: Tools can misunderstand the data, which may cause wrong results. Relying too much on results from monitoring tools may create an inaccurate view of the network.

3. **Complexity**: Network monitoring can be complex, and even with specialized tools, it may require a lot of time and knowledge to do it properly.

4. **Security risks**: Monitoring tools gather information along the whole network and store it in one place. This data gathering may create a security risk if not appropriately secured. An attacker can gain access to sensitive data by compromising the monitoring solution.

Several factors can cause problems with network monitoring. Some of the common issues that can impact network monitoring include:

1. **Network congestion**: When a connection link or a network device is saturated, some packets may be delayed, dropped, or retransmitted. In the same way, as congestion can appear in the monitored network, it may also appear in the monitoring network. Congestion can lead to gaps in monitoring data.

2. **Malicious activities**: Malware applications can generate artificial data to confuse, overload or manipulate monitoring tools.

3. **Missing data**: Inadequate monitoring settings can lead to a situation when an important part of the network is hidden from an administrator. Incomplete or ineffective monitoring can limit the ability of network administrators' ability to see what is happening in the network.

4. **Asymmetric routing**: With asymmetric routing, not all packets between the source and the destination are transmitted over the same route. This may make some packets visible in a different monitoring point than others within the same communication.

To mitigate problems with network monitoring, network administrators should carefully plan their network monitoring strategy and use the right tools and techniques while minimizing overloading and complexity. Regularly reviewing and updating network monitoring policies is also important to ensure they remain effective.

Each technology for network monitoring has its strengths and weaknesses, and the choice of which technology to use will depend on the specific needs and goals of the organization. Some factors that may be considered when choosing a network monitoring technology include the size and complexity of the network, whether the monitoring infrastructure may interact with the monitored network, the types of devices and protocols being used, the level of detail required for monitoring and analysis, and the budget available for monitoring tools and resources.

2.3 Analyzing Network Packets

As mentioned in the previous section, multiple data sources are used for network monitoring. However, this thesis focuses on methods specific to analyzing the network packets. For this reason, only the processing of network packets is described more deeply in this section. The network packets have been chosen because this data source does not require any changes to the end stations and only minimal changes to the monitored networks - duplicating traffic to monitoring points. There is also no need to inject artificial traffic into the monitored network, meaning this approach can not negatively affect end users or network applications. By analyzing every packet that is transmitted through the network, a highly detailed view of network activities is created. The limitation is that the methods cannot access information located in end-stations or network devices.

2.3.1 Network Packets

Network monitoring from packets involves capturing and analyzing packets to gain insights into network activities. In addition to application data, network packets contain protocols from the lower network layers, which may contain important information - e.g., source and destination IP address. To analyze network packets, administrators typically use one of the three following approaches:

1. **Manual or in-depth analysis** of packets with packet capture and analysis tools, also known as packet sniffers. These tools allow administrators to inspect each packet, decompose the packet's data into used protocols and display the value of each packet's byte in a human-readable format [8]. Administrators use these tools to troubleshoot

network issues, analyze network protocols, and identify potential security threats. The most popular tool for packet analysis used nowadays is Wireshark, shown in Figure 2.3. Wireshark is an open-source tool with a huge community that supports over 3,000 protocols, making it the most powerful analysis tool.

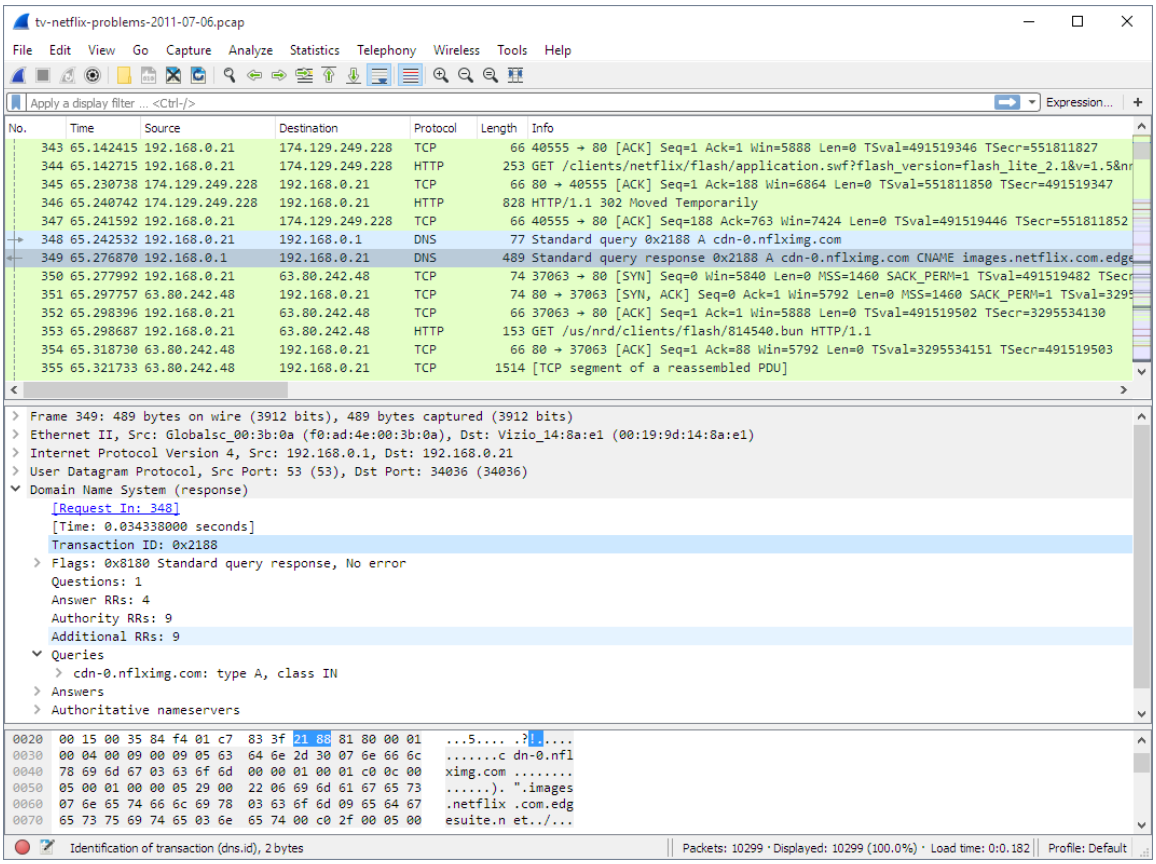

Figure 2.3: The GUI of the Wireshark tool showing the content of the selected packet.

The main disadvantage of manual analysis is that it is a very time-consuming activity [142]. It is only possible to investigate some things happening inside the network with this approach. It is mainly used when all other approaches fail. Another disadvantage is that this activity requires a deep understanding of computer networks. On the other hand, with proper knowledge and enough time, it is the most powerful option.

2. **Exploration or top-level analysis** of network traffic means that instead of focusing on individual packets, only statistical information from network packets is presented to an administrator [116, 44]. For example, the number of transferred bytes or top talkers inside the network. Even though this kind of analysis is often possible to do with packet sniffer tools, those tools are missing automation and integration features [89]. One example of such a tool is the SolarWinds company's Network Packet Analyzer, as shown in Figure 2.4.

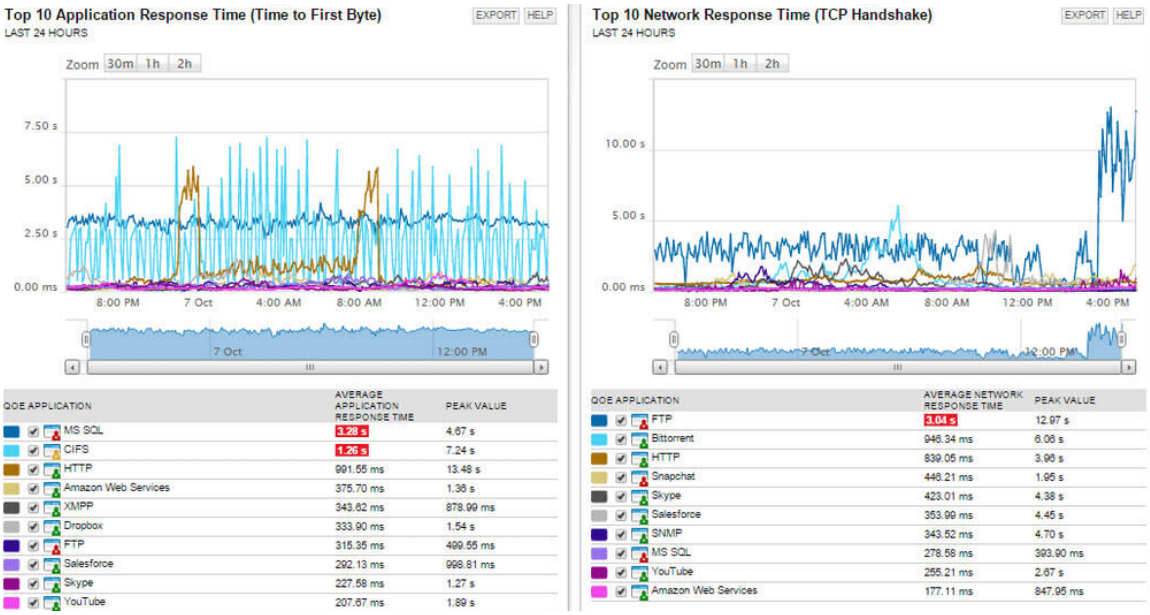

Figure 2.4: The GUI of the Network Packet Analyzer that shows the 10 worst applications based on their network performance statistics.

The primary advantage of this approach is the administrator's ability to obtain a quick overview of the network's activity. Instead of presenting administrators with excessive details, only summarized information is provided. This approach enables continuous monitoring of networks of various sizes, but it comes at the expense of reduced granularity. Typically, an administrator can identify an issue through exploratory analysis, but additional manual investigation is necessary to determine the cause. The disadvantage of this method is that since it relies solely on aggregated data, it is critical to identify problems that can only be discovered through detailed analysis.

3. **Automatic analysis** analyzes all the packets and, based on decision algorithms, automatically reports to administrators what is happening inside the network [103]. These reports include decreasing performance, detecting security policy violations, or identifying a potential problem. An example of an automatic analysis tool is Suricata, an intrusion detection system (IDS) that can detect security incidents. In Figure 2.5, it is possible to see a list of security events.

2018-06-30 15:58:19 6 minutes ago	S: 14.157.159.75 D: 10.44.100.50	ET SCAN Suspicious inbound to MSSQL port 1433
2018-06-30 15:54:49 10 minutes ago	S: 107.170.255.53 D: 10.44.100.50	ET CINS Active Threat Intelligence Poor Reputation IP group 98
2018-06-30 15:54:25 10 minutes ago	S: 218.60.67.79 D: 10.44.100.50	ET SCAN Suspicious inbound to mySQL port 3306
2018-06-30 15:53:23 11 minutes ago	S: 185.8.49.228 D: 10.44.100.50	ET COMPROMISED Known Compromised or Hostile Host Traffic group 10
2018-06-30 15:53:19 11 minutes ago	S: 211.239.113.60 D: 10.44.100.50	ET SCAN Suspicious inbound to MSSQL port 1433
2018-06-30 15:51:02 14 minutes ago	S: 10.44.100.235 D: 233.23.34.71	ET P2P BitTorrent DHT ping request

Figure 2.5: List of detected security events from the Suricata tool.

Compared to previous approaches, the result of the automatic analysis is only information that fulfills some criteria. On the other hand, previous approaches give administrators much data, and it is up to the administrator to filter out what is important and what is not. This automatic processing saves a lot of time and effort for network administrators. Because the analysis capability is included inside the tools, administrators also have lower knowledge requirements. The most significant benefit of this approach is that it is possible to permanently monitor huge networks with many devices and connections. However, the tools may falsely assume that everything works as expected when nothing is detected. The problem is that tools can detect only situations they were made for, and everything else is undetectable.

2.3.2 Network Packets Analysis

Before it is possible to make an analysis and come to a conclusion from network packets, it is necessary to have access to the packets, capture them, extract information from them, and then it is possible to analyze them. All these actions are shortly described in this subsection.

Getting Access to Data

There are several common ways to access network packets [111], as shown in Figure 2.6. Software tapping is the easiest one that does not require any additional hardware in the network. With software tapping, packets are copied inside the end devices to applications using software means. Even though it is the easiest way, with this approach, only packets sent from or to the device where the software tapping is happening are available. Another approach is to create copies of network packets on network devices (switches and routers) and send those copies to a special monitoring device. This activity is called port mirroring. With this approach, all traffic going through the network device is copied, and it is possible to access a large portion of network traffic. The port mirroring must be supported on network devices. Even with full support, the port mirroring is unreliable as copying the traffic has very low priority on devices. When the device is highly loaded, many packets are not copied. Therefore it is impossible to analyze them. The last common way of accessing network packets is by using a special network device called a tap. This device is placed on the network link (no matter the physical technology used) and copies all the packets to the special device. With this approach, copying the traffic does not affect the monitored network itself; accessing all packets transferred over the tapped links is possible. The disadvantage is that special tapping devices must be put inside the network.

Capturing the Data

After having access to network packets gathered either by software or hardware copying, it is necessary to capture the data [144]. The most popular library used to capture and process packets from the network is libpcap[2]. It provides a platform-independent API for capturing and filtering packets and can be used with various programming languages, including C, C++, and Python. Libpcap also allows storing data for future processing by using the PCAP format. PCAP (short for packet capture) is a binary format that stores all packet data together with arrival timestamps and the length of each packet.

[2]https://www.tcpdump.org/

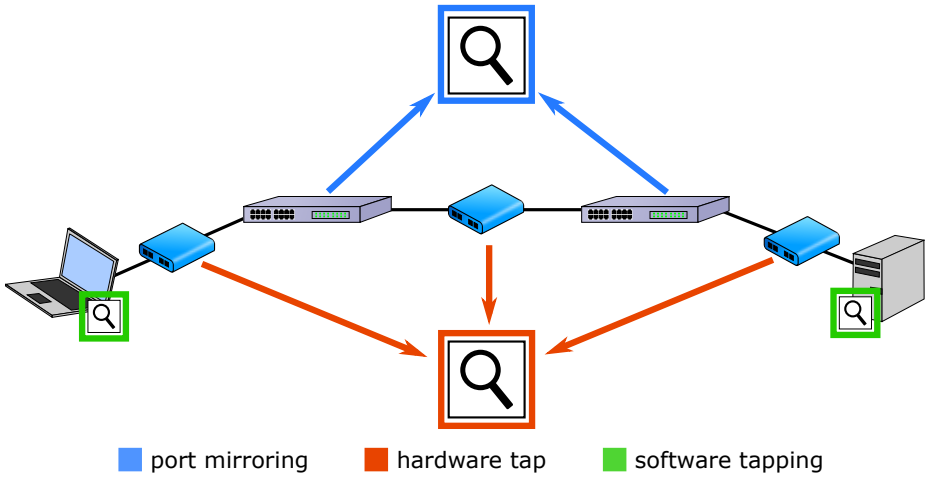

Figure 2.6: Three most common ways of accessing the network packets. With software tapping, it is possible to access packets on end devices. With port mirroring, the packets are copied on network devices, and with hardware taps, packets are copied directly from the physical links.

Parsing the Packets

To understand packets that are stored in a binary format, it is necessary to recognize the structure of the packets and extract relevant information. For packet recognition, a protocol parser is used. A protocol parser is a software module that analyzes the contents of a packet based on its protocol. It extracts data from the packet header and payload and interprets it according to the protocol specification, allowing for the meaningful analysis of the packet. As the packets consist of multiple encapsulated protocols, the parser needs to parse all of them. Because each protocol differs, the protocol parser must contain specifications for hundreds or thousands of protocols to support all the protocol packets. Many research activities also focus on the problem of creating new protocol parsers, so if the parser for the required protocol is not available, it is possible to create a new one [42, 40, 11, 126].

Packets Data Analysis

After analyzing the packets, the data can be used for various purposes, such as troubleshooting network issues, optimizing network performance, or detecting network security threats. For example, packet analysis can help identify network bottlenecks or configuration errors, improving network performance. It can also help detect and isolate malicious traffic or unusual network activity, allowing faster response to potential security incidents. Additionally, packet analysis can assist in compliance monitoring and audits and provide insights into network usage and application behavior.

Wireshark is an example of a tool that can capture, parse and analyze data. Wireshark provides a user-friendly interface for viewing and filtering packet captures and supports many protocols and network interfaces. A command-line version of Wireshark called TShark exists that provides the same functionality and is more suitable for pipelines with automatic processing. Although Wireshark understands application error codes and some TCP errors, Wireshark is not a perfect tool for diagnostics as it is missing complex diagnostics functions [89].

2.4 Network Diagnostics

Computer network diagnostics is a central aspect of network management [69]. It is a process of identifying and resolving problems within a computer network. An example of such a problem can be slow connectivity, unavailable service, or malware inside a network. Network diagnostics history started in 1967 when a Mechanical Failure Prevention Group was formed that focused on hardware errors on network devices [94]. Network diagnostics aims to ensure that the network is functioning optimally. When there is a problem, the diagnostics must resolve it quickly and efficiently. Because the correct functionality of computer networks is critical, administrators must pay adequate attention to network diagnostics. Diagnostic is used in various areas where computer networks are utilized: IoT [73, 47, 89, 112, 56], automotive industry [9], mobile networks [98, 20], data centers [141, 125, 81, 145, 119], home networks [58, 19, 109], Internet sites [21, 59], virtual machines [131], application microservices [120, 17], software errors [14, 35, 146, 138], network links [93, 108, 52, 71, 78], IPTV [75, 80], transportation [91], or security [72, 29, 90]. Several types of people do network diagnostics: technical support operators, engineers managing servers and services, and network designers [68]. In this work, we do not distinguish these roles; when using the term administrator, we mean any of these roles.

Before continuing with describing the network diagnostics, it is essential to understand the basic terminology [69, 33, 106]:

- **Event** - An exceptional condition that occurred within the managed network.
- **Faults** (or root causes) - The type of events that can cause other events.
- **Error** (or failure) - consequence of a fault. However, not each fault needs to cause an error. The error refers to a problem in the network that affects its performance, reliability, or availability.
- **Symptom** - A symptom refers to the external manifestations of errors, which can either be directly observed as the errors themselves or be indicated by externally visible signals that an error has occurred. The latter can, for example, include alarms raised by anomaly detectors.
- **Root Cause Analysis** - Also used by the abbreviation RCA is a process used to identify an error's underlying cause(s).

2.4.1 Importance of Network Diagnostic

Network diagnostics is important because network errors are inevitable [69, 118]. From the downtime statistics of several cloud services [38], we can see that one hour of an outage caused by an error can cost up to 336,000 USD. Of course, this cost is valid for large service providers with many customers, and therefore, the cost of downtime for smaller and more common networks would be much lower. At the same time as having costly errors, more organizations in all industry sectors recognize they are evolving into technology and data companies [106]. By identifying and addressing issues early on, network diagnostics can prevent more severe and costly problems from occurring later. Additionally, network diagnostics can also help optimize network performance, resulting in cost savings through more efficient use of network resources. These facts lead to the fact that network diagnostics cannot be underestimated.

Network diagnostics is not an easy task to do [7, 69]. As mentioned in previous sections, a huge variety of errors may happen inside the networks. As networks become more complex and diverse with the advent of new technologies and types of devices, diagnosing and fixing issues becomes more challenging. Additionally, the increasing use of cloud-based and virtualized infrastructures further complicates the diagnostic process, as it becomes harder to determine the root cause of issues across disparate systems and environments. Furthermore, the large volume of data generated by modern networks makes it difficult to identify patterns and trends that could indicate underlying issues. All these factors make network diagnostics a continually evolving and challenging field. Figure 2.7 shows a chart representing the time required to solve one error by an administrator. The chart was made based on a survey in which 58 administrators of various network sizes were included [139].

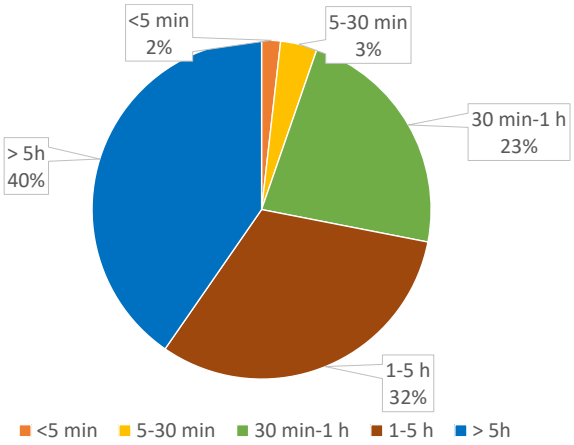

Figure 2.7: Time consumed to solve one error by an administrator.

Performing diagnostics requires a deep understanding of network protocols, tools, and technologies. It involves analyzing and interpreting various data to find the source of errors. Additionally, errors may not constantly occur in the same place, making it challenging to find the exact location and source of the fault. In some cases, the error may result from a combination of factors, making it even more challenging to isolate the problem. Without sufficient knowledge and experience, diagnosing and resolving network problems can be challenging. The complexity of networks, the advent of new technologies, and even other problems not mentioned here lead to a situation where network diagnostics is still an open research problem.

2.4.2 Errors in Computer Networks

The network diagnostics focus on errors, primarily trying to find a way to eliminate them. Therefore, it is necessary to understand what those errors are, what they may cause, and what sources of errors exist. Several types of errors can occur in computer networks [7]:

1. **Availability errors** - These errors occur when a resource, like a device, service, or link, is unavailable or has inconsistent behavior (e.g., frequent disconnections or dropouts) [108, 57, 146, 28]. These errors prevent users from accessing and using the services of those resources. An example can be when a user cannot visit a specific web page.

2. **Performance errors** – Network resources are not just up and down but also can have performance issues [12, 10, 19, 78, 109]. The performance issue means a resource is overutilized or experiencing bottlenecks, resulting in slow or degraded performance.

3. **Security errors** – In this case, users can typically access the resources with an expected performance (although some security errors can co-occur with other availability or performance errors). However, the resources may be compromised or attacked, resulting in data breaches [103, 104, 25, 101]. An example of such an error is malware infection spreading inside the network [90].

All these types of errors can cause a variety of damages. When a service is unavailable or has decreased performance, it may lead to a loss of productivity or revenue. Some errors can create vulnerabilities that hackers can exploit to gain unauthorized access to a system or network. This may lead to data leaks, espionage, or loss of data.

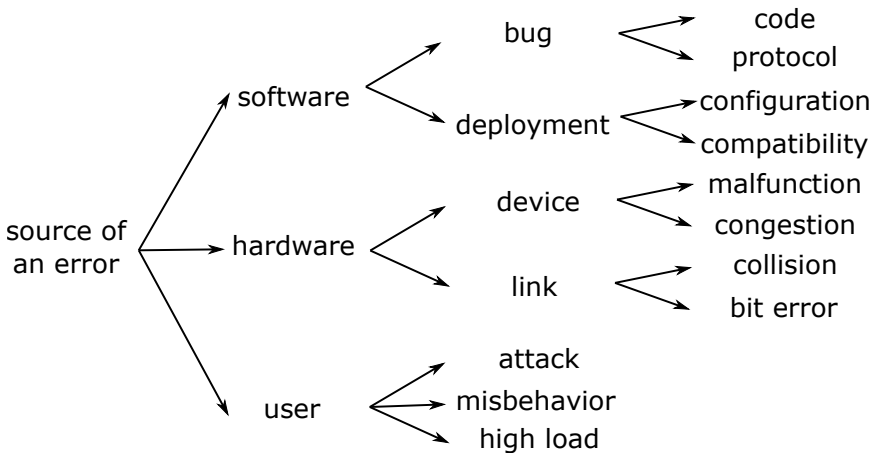

Figure 2.8: Classification of errors based on their sources.

Network diagnostic is challenging because an error may appear from many sources. The error can be created by pulling out the wrong cable, applying a new configuration to the network firewall, or with a bug in the application code. The following text describes the complete classification that can also be seen in Figure 2.8:

1. **Software** – These errors are caused by software, like user applications or network device operating systems.

   (a) **Bug** – These errors are related to an application not being implemented as expected (by users or standards). It is often hard to fix these errors as applications are usually not developed by companies running them.

      i. **Code** – Even if an application is well designed and tested, a bug may happen anytime, resulting in an application crash or unpredictable behavior. For example, there can be a segmentation fault when processing input data or an unexpected sequence of valid requests.

      ii. **Protocol** – When two different applications communicate, they must use a network protocol to understand each other. They are usually well-defined and implemented, but with new protocols, some misunderstanding may appear as each application can understand some protocol details differently, resulting in communication failures between devices.

(b) **Deployment** – Besides a correct implementation, an application must be set up to work correctly. However, many applications have a lot of parameters and complicated configurations, which can be hard to manage.

i. **Configuration** – These errors occur when there is a problem with the network or application configuration problem, such as incorrect IP addresses or DNS configuration. An application or network device may not work as expected with an incorrect configuration. For example, the user may not access a specific website.

ii. **Compatibility** – These errors occur with applications that have been used for a very long time, and during that time, new versions have been released. As new versions bring new features and replace the old ones, the new versions may not correctly handle requests made initially for older versions.

2. **Hardware** – These errors are caused by elements not caused by the wrong software.

(a) **Device** – These include end-user devices like computers, servers, or phones, network devices such as routers or switches, and middleboxes like firewalls or load balancers.

i. **Malfunction** - A malfunction of a network device can occur due to various reasons such as high temperature, high humidity, power fluctuations, and physical damage to the device.

ii. **Congestion** - Congestion occurs when the network becomes saturated with traffic, causing delays, packet loss, and degraded network performance.

(b) **Link** - These errors occur when data is transmitted over the network links (both wired and wireless).

i. **Collision** - Collisions occur when two or more devices simultaneously attempt to transmit data over the same link. This can result in the loss of data packets.

ii. **Bit error** – These errors occur when a bit in a data packet is corrupted during transmission. Issues with the transmission medium, interference, signal attenuation, or loss of synchronization can cause this.

3. **User** – Errors in this section are caused not by technical elements but by humans.

(a) **Attack** – This includes errors when someone tries to attack network resources on purpose. Attacks include both manual and automatic attacks made by malware, for example.

(b) **Misbehavior** – Users can inadvertently cause errors by performing actions that are not allowed or failing to follow proper procedures, such as running warez software that causes a DoS attack on an internal server.

(c) **High load** – Even when users use everything as they should, an error may occur. When many users download huge files simultaneously, the network traffic may exceed its capacity and cause delays, packet loss, or other errors.

2.4.3 Steps of Network Diagnostics

Network diagnostics can be performed manually or using automated tools and involve both hardware and software components of the network. All these approaches involve using various techniques to identify and troubleshoot network issues. Identifying and troubleshooting

issues helps administrators find the underlying causes of problems, determine what factors are contributing to the problem, and help fix the issues and restore the network's functionality. Additionally, diagnostics can help administrators to identify potential risks and vulnerabilities in the network, enabling them to proactively address these issues and prevent them from causing future problems.

Both manually and automatically performed diagnostics involves detecting and analyzing errors in a network to determine the cause and location of the issue and then implementing corrective measures to resolve the problem. During the whole process, it is necessary to define how the diagnostics should have access to the data it will analyze. The options and techniques for collecting the data were already described in the previous section, "Network Monitoring". The process is usually defined as three independent steps [99, 69]:

1. **Error detection** – The first step identifies whether an error has occurred. This process involves recognizing an issue with the network or application performance. Many times, errors are detected directly by end-users.

2. **Error localization** (or root cause analysis) - Once an error is detected, error localization is used to find the location or source of an error, such as a network segment, specific device, or application. The goal of error localization is to narrow down the scope of the problem so that corrective action can be taken more quickly and effectively.

3. **Error correction** – The correction tries to fix the detected error to restore the expected functionality. Fixing can involve various actions, such as reconfiguring network settings, replacing faulty hardware components, updating software or firmware, or applying patches or security fixes. The goal is to eliminate the underlying cause of the error and ensure that the network is functioning correctly.

This dissertation thesis focuses on the first two diagnostic steps – error detection and localization. These two steps can further be executed in the following substeps (the description is similar to description of analyzing network packets in the subsection 2.3.1):

1. **Top-level analysis**: This involves looking at the network as a whole and identifying any significant issues or areas that require further investigation.

2. **Automatic tools**: Once the top-level analysis is complete, automatic tools can gather more detailed information about specific areas of the network. These tools can help to identify specific problems and provide detailed data for further analysis. For example, by analyzing all SNMP messages, the tool may conclude that the problem is with the user's authentication.

3. **In-depth analysis**: Network diagnostics often involve a more in-depth analysis of the data collected in the previous steps. This analysis can involve manual analysis of log files, packet captures, or other data sources to identify the root cause of the problem and develop a plan for correction.

Usually, the root cause analysis starts with step one (top-level analysis) and continues with the next steps. However, each of these steps is optional. Based on the administrator's experience, some steps can be avoided or repeated multiple times until errors are fully diagnosed and resolved.

2.4.4 Automatic Diagnostic Tool

With the current complexity of network diagnostics, it is no longer possible to do the diagnostics manually, but it requires a higher level of automation [106]. Automatic diagnostic tools can support the administrator with the knowledge that he or she may not know. Because the complexity of networks is very high, the administrator may not have enough knowledge to locate the problem. Another benefit is speeding up the whole diagnostic process [142]. Even if the administrator has the required knowledge, sometimes finding the correct cause of the error is very time-consuming. Therefore, even if the administrator is not reliant on such a tool, it is still helpful because the administrator spends less time diagnosing the problem. Reducing the time and the knowledge required by administrators helps organizations with operational costs [99].

An automatic diagnostic tool is designed to help network administrators and technicians identify errors affecting network performance, availability, and security. The tools usually do not cover all steps of the diagnostic process. The goal of tools that focus on error detection is to find unusual activities or states inside the network [100, 120, 64, 79, 18, 83, 54, 63, 109]. These tools are often called anomaly detection systems [83, 24, 16, 72, 82]. Another set of tools focuses on error localization. These tools aim to provide a comprehensive view of network errors so that administrators can resolve errors quickly and effectively. The error correction tools are focused tools that can directly interact with monitored infrastructure and fix errors so the administrator can only verify the results [127, 50]. This dissertation thesis focuses on automatic tools that deal with error localization, and the rest of this chapter covers only these types of tools.

As already mentioned, automatic tools aim to provide important and valuable information to administrators to help them diagnose errors [100, 69]. To be able to do this, the knowledge that imitates the knowledge of a human required to do the analysis needs to be integrated into the automatic tools. The knowledge can be either surface (from experience) or deep (from understanding the system behavior) [69]. In addition to the knowledge base, the tools also require access to the input data and sometimes additional input information (or parameters) from administrators. The existing solutions are not ideal because they often do not fulfill all the requirements expected by administrators.

There are several types of requirements placed for a diagnostic system:

1. **Capabilities** – The ideal system should be able to detect, identify and correct each error. However, this is not always possible as a passive-based system cannot interact with monitored networks, and therefore, it is impossible to execute correction steps to fix errors. The system should be able to work with known and unknown errors, as it is not always easy to specify all possible errors while building the system [64, 34, 72]. The last important capability is that in addition to modeling the knowledge required for diagnostics, the system must specify how to work and process the input data, as this aspect is sometimes ignored [133, 34].

2. **Maximum autonomy** – Another aspect of the diagnostic system is automatically diagnosing errors with minimum user interactions [100, 32, 105] and changes made to the monitored networks [4, 115, 48]. The system should only work with provided data without having access to historical (training) data [6, 113, 89, 54] and access to active actions [1, 12, 3, 39]. The system should also work without the necessity of implementing new protocols [141], new applications for the monitored devices [1],

or without needed access to other devices and networks to perform the diagnostics [31, 58, 131, 57].

3. **Variety types of errors** – A diagnostic system must not focus on a single type of error [106, 127, 65, 130]. Ideally, all types of errors must be supported, no matter where they are located (like end stations, middleboxes, or network devices) [92, 67], and all TCP layers must be covered [142, 69]. The system should be able to distinguish unusual situations but does not need to be a problem [58]. Diagnosing techniques must not be focused on a single type of network [54]. All types of networks must be supported, for example, Internet service providers, data centers [141, 125, 81, 145, 119], industrial networks [22, 74, 5, 89, 112], home networks [58, 19], or enterprise.

4. **Good knowledge representation** – In a real-world environment, diagnostic systems often fail with how the system works with the knowledge base and how the outputs are linked to this knowledge [15]. For example, no matter what the result of the system is, results always need to be justifiable. Administrators need to know how the system came to such a result, what it means, and how to use that information [106]. If the knowledge base is incomplete, administrators must be able to update it easily – add support for new errors or replace descriptions of inaccurate errors. It is also essential that the system produces the same results for the same errors and input data and that the behavior is not random.

2.4.5 Classification of Diagnostic Techniques

Network diagnostics lacks a single universal technique due to the variety of network environments and the complexity involved. The choice of diagnostic technique depends on the specific requirements, the complexity of the network, the resources available, and the desired level of accuracy in diagnosing and resolving network problems. For example, sometimes it is impossible to use an active approach, and only passive methods are available. Another example is when diagnostics are only possible from application log files. In this section, we provide a classification of diagnostic techniques based on several criteria. It is important to note that even though some techniques may be used more than others, the best technique is never in any of these selected criteria. In reality, diagnostic tools often combine several techniques, such as working with multiple data types simultaneously.

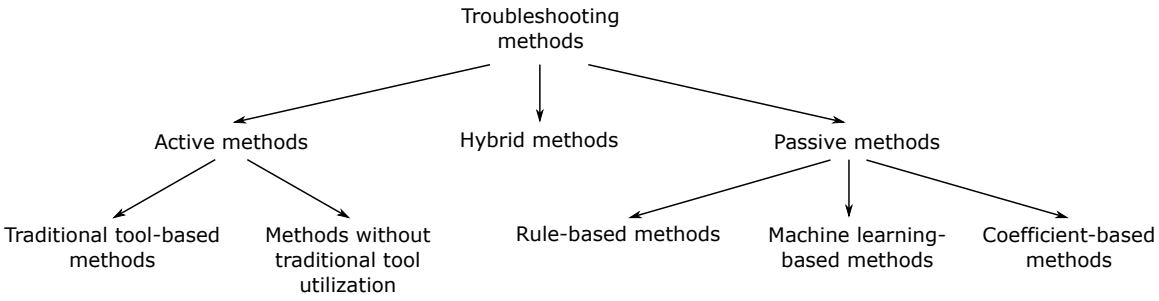

Figure 2.9: Classification of troubleshooting approaches as given in [117].

Several papers put some hierarchy into the classification criteria [69, 117]. Figure 2.9 shows an example of such a hierarchy. In most cases, however, these hierarchies do not make sense. The figure shows that passive methods can be further classified into rule-based methods. However, active and hybrid methods can also be rule-based. For this reason,

the criteria in this section are not organized hierarchically as in other papers. Instead, the N-dimensional approach is used, where one class is assigned from each dimension, resulting in N independent classes. The classification criteria for network diagnosis and troubleshooting methods include various factors such as knowledge base, extensibility, data sources, working with training data, impact on the monitored network, automation capabilities, execution location, error types, implementation algorithms, use cases, cooperation, diagnosis time, and diagnosis steps. Few existing techniques are then evaluated against these criteria.

Data Sources

The most important factor distinguishing different diagnostic methods is the type of data they are working with. The most common data types are log files and network packets. It is also widespread that the methods are doing a fusion of information - combining information of different types to perform diagnostics. The following Table 2.1 shows which data types are papers that were analyzed in this thesis using.

papers (references)	Log files	Network packets	Network flows	Alarms	Syslog messages	SNMP	Resource utilization	System traces	Routing protocols	Device configuration	Performance info	Source codes
1, 20, 23, 39, 44, 47, 56, 77, 106, 115	x											
10	x	x	x									
29	x			x								
30					x							
41						x						
48, 67		x										
54, 72			x									
75	x						x					
83								x				
84, 102									x			
87		x								x		
93				x	x	x		x		x		
96, 136											x	
101	x		x			x						
135				x	x	x			x	x	x	
120	x											x

Table 2.1: Classification of analyzed papers based on data sources the papers are using.

Implementation Algorithms

The second most important attribute that categorizes each diagnostic method is the used implementation for the inference model. A previously published paper by Solé Marc et al. titled „Survey on models and techniques for root-cause analysis“ [106] provides an exemplary analysis of different diagnostic models, categorizes them, elucidates the learning algorithms employed, and presents an assessment of their complexity. Given the thoroughness and quality of their work, the reference to their paper is used for the classification of related work in this category. Figure 2.10, sourced from their publication, visually presents a selection of widely recognized models in this field.

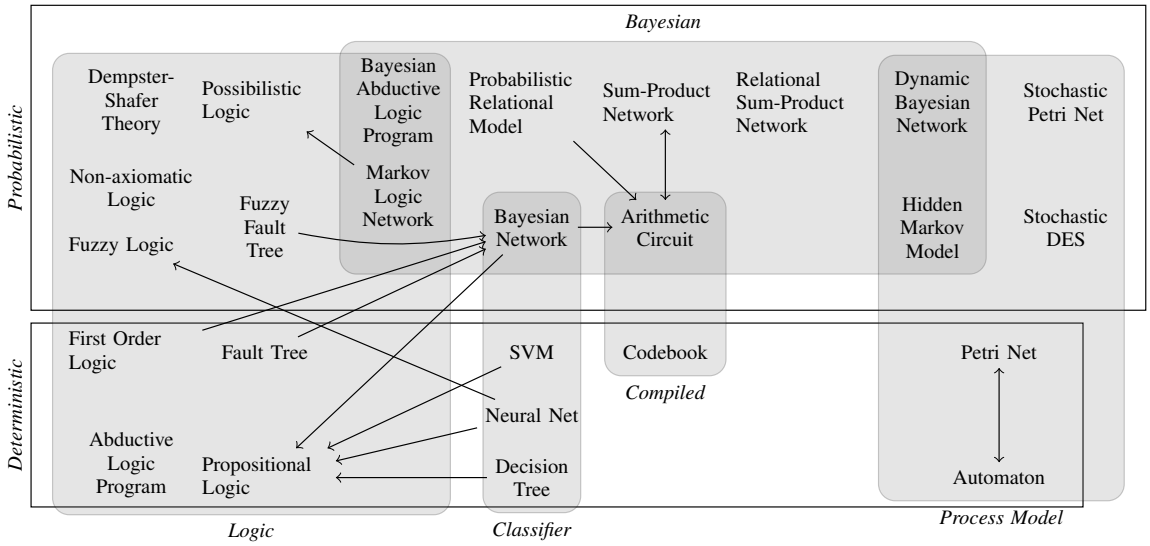

Figure 2.10: Classification of interference models used for diagnostics [106].

It is very important to notice that even when machine learning classifiers are an attractive topic (both in research and among administrators), they do not dominate the area of diagnostic errors. As correctly stated in the mentioned paper, machine learning algorithms only return a result (for example, what is wrong) without any additional explanation. Another problem is that because these techniques are not based on logical rules, it is very hard to integrate the domain knowledge of expert users into the final models.

Affecting Monitored Network

Another important criterion for differentiating diagnostic methods lies in determining whether the method imposes additional traffic on the network being monitored. This criterion serves as a crucial factor in evaluating the impact and feasibility of various diagnostic approaches.

The following options are available:

1. **Passive** - These methods do not create additional traffic and work only with already available data [129, 140, 87, 23].

2. **Active** - The methods send artificial traffic into the network to gather diagnostics data. This traffic is then either captured and analyzed in different network locations, or the traffic causes network devices to send answers that can be later analyzed. In addition to the complex diagnostic methods [73, 43, 105, 4, 93, 12, 3, 71, 114], this category includes traditional tools like ping, traceroute, or tcpdump.

3. **Hybrid** - This category contains methods that combine passive and active approaches into one solution. The benefit of the combination can either be the possibility to extend the information available from the passive approach by actively gathering additional information [105, 115, 49], or the benefit can be that the errors that are detected by the passive approach can be localized with active traffic [48, 39].

Knowledge Base

In order to diagnose network errors, diagnostic methods utilize a knowledge base that outlines troubleshooting procedures for specific errors. The depth of this knowledge base can vary. Methods with a **deep knowledge** base possess a comprehensive understanding of network protocols and operations [76, 105, 65]. In the event of an error, they can dive into the complexities and identify the root cause for effective resolution. On the contrary, methods with a **shallow knowledge** base lack the underlying understanding of why errors occur. Their knowledge is primarily based on past experiences [22, 74, 77, 99, 87, 136]. For instance, if the method determines that a specific error with a DNS server is due to misconfiguration, it will consistently provide this solution for similar errors in the future. Building knowledge bases from past experiences is relatively simpler, as it involves providing solved examples rather than comprehending the underlying details. However, methods relying on shallow knowledge may yield inaccurate results as they lack a deep understanding. Additionally, models based on past experiences may struggle to effectively address new, previously unseen errors.

Extendability

Some methods have a closed knowledge base that is very hard to extend, usually only by changing the source code. This closed approach has a disadvantage that an administrator is **unable to extend**, as a typical network administrator is usually not a programmer [75, 60, 61, 80]. The second method category has an open knowledge base that can be **easily extendable** to cover new errors. The extensibility can have the form of allowing a new set of rules in a user-friendly format [20, 76] or the form of providing sample traces [90].

Working with Training Data

Another classification criterion is whether or not training data that contains errors that will be diagnosed in the future will need to be provided. Providing this data can often be problematic or impossible. For example, some failures may be difficult to simulate in order to create a training data snapshot for learning diagnostic methods. Three categories exist:

- **No data** at all - Methods like static rule sets are created manually by experts with domain knowledge [134, 32]. To cover a specific error, an expert needs only to describe how the problem will be detected and diagnosed. The benefit is that diagnosing errors that have never happened before is possible. The disadvantage is that domain expertise is required.

- Using data **during development** - Methods of this type require data only once when the methods are being developed [110, 63, 53]. The result of the deployment phase is a generic model, which is then deployed in different network configurations. The disadvantage of the generic and universal models is that models do not respect and adapt to individual networks. The advantage is that there are lower requirements for domain knowledge, however, at the cost that data samples must be provided.

- Using data **during deployment** - Sometimes, using a pre-created generic model is impossible. Instead, the model must be created after the diagnostic method has been

deployed into the monitored network [74, 64, 72]. For example, when detecting a performance issue, it is necessary to know the typical performance in the network. The advantage is that the detections are made based on specific network configurations. However, at the cost of some time before the diagnostic method can start work - either a specific situation must occur inside the monitored network, or a minimum amount of data must be processed.

Automation

From the user's perspective, a very important classification criterion is the level of automation. Some of the methods are **fully automatic**. They have been developed so that during the monitoring network deployment, administrators are just getting the results without any user input [131, 97]. Other methods are **semi-automatic** and require some degree of user interaction. The word semi means that the methods are mainly fully automatic. However, there are some moments when they require manual user input. For example, methods may require the user to manually **mark the input data**, so the method can learn how to analyze the network [2, 55, 21, 89, 1, 53]. Another possibility is that the method asks users to **answer questions** either during the learning phase [34] or during the diagnostic process [100, 32, 105, 58].

Location of Execution

Network diagnostics can be executed at various points within a computer network. Here are some typical locations where network diagnostics can be performed:

1. **Central server** - Data required for diagnostics (e.g., log files or NetFlow records) are sent to a separate machine where the diagnostic process is executed [57, 95, 79]. This is the most common variant because the process is not affecting the monitored network, and it is easy to gather the data from multiple locations.

2. **Cloud** - Similar to the previous variant, but in this case, the diagnostic process is executed outside the monitored network in the external cloud environment [131, 41, 128]. Because the cloud can have more resources, implementing the diagnostic process can have higher performance demand. The most significant disadvantage is that sending data outside of the monitored network can lead to a leakage of sensitive information.

3. **Network device** - Some diagnostic activities can be directly executed on network devices, e.g., routers [109]. In reality, performing the diagnostic process directly on network devices can negatively affect the performance of the primary device activities like routing packets. Therefore, this option is barely deployed in real networks.

4. **End device** - The benefit of diagnosing directly on end devices is that transferring data for diagnostics to any other device is unnecessary as they are already available where the diagnostic is happening. In addition to regular computers [58], methods can also work on mobile phones [135].

Error Types

Diagnostic methods can be categorized according to the specific types of errors they are designed to detect [117]. Depending on their focus, certain methods specialize in diag-

nosing **availability** errors [7], and **performance** errors [86], while others target **security** issues [103, 104] or **application** errors [55]. This classification allows for more targeted and efficient troubleshooting, as different diagnostic methods address distinct aspects of network functionality and can provide valuable insights into specific error types. By tailoring the diagnostic approach to the specific error domain, methods can effectively pinpoint and resolve issues, ensuring optimal network performance, security, and application functionality.

Use-cases

Diagnostic methods can be specialized, targeting **single specific use-cases** such as detecting changes in Internet paths using BGP protocol [49], detecting problems in IoT networks [47, 22, 74], or identifying congestion issues within TCP protocol [102, 85, 124, 65]. Conversely, there are also **versatile methods that serve multiple purposes** and exhibit adaptability across various scenarios [76, 133, 96]. These multifunctional diagnostic approaches possess the flexibility to address a wide range of network challenges, providing comprehensive insights and actionable information for diverse network conditions and requirements.

Cooperation

Diagnostic methods can operate individually in **isolation** [121, 26, 91] or in **collaboration**. The collaboration can be based on the communication between multiple instances of the method or on the communication with a central server. Collaboration can expand the knowledge base by merging knowledge from diverse sources [1], or it is possible to use communication channels for detecting anomalies and problems [60, 19, 57]. However comparing with methods that work in isolation, the collaboration requires functional communication with others even when an error occurs.

Time of Diagnostics

Usually, the diagnostic process starts when an error is detected. The error is then analyzed and solved to restore the standard network functionality. Methods that work this way are **reactive** [72, 80, 10]. On the other hand, there are some scenarios when it is possible to troubleshoot a problem before it leads to an error causing availability or performance issues. For example, when the translation time of the DNS server starts to increase, it is possible to check the server's utilization or utilization of network links and devices even before the users notice a worse user experience. The methods that can diagnose a problem before it starts to cause errors are called **proactive** [52, 3, 61].

Diagnostic Steps

Diagnostic methods typically do not include all the diagnostic steps (outlined in section 2.4.3) but focus only on the first one or first two of the three steps:

1. **Detection**: These methods primarily focus on detecting errors within a network. Their main objective is to identify the presence of any anomalies, deviations, or issues that might be affecting the network's performance or functionality. By monitoring various network parameters, traffic patterns, or system behavior, these methods can recognize the existence of errors [100, 129, 120, 109].

2. **Localization**: In addition to error detection, certain diagnostic methods go a step further and aim to localize the source of the detected errors. By leveraging advanced techniques such as packet analysis, path tracing, or network topology mapping, these methods are trying to find a location or component within the network infrastructure where the errors originate. This localization helps narrow down the troubleshooting process and facilitates targeted corrective actions [73, 81].

3. **Correction**: Error recovery represents the most difficult phase of the diagnostic process. While many diagnostic methods excel at error detection and localization, only a select few can calculate and suggest recovery options to rectify the localized errors. These advanced methods identify the source of the errors and propose potential solutions or mitigation strategies to restore the network's normal operation. This involves evaluating alternative configurations, rerouting network traffic, adjusting settings, or implementing corrective measures at the identified error source [127, 50].

Focusing only on a subset of steps allows us to use more specialized approaches and algorithms that, for example, may work great for error detection but not very well for error correction. In some cases, it makes sense to combine several independent works into one larger solution in which one work is focused on error detection, and the detected errors are then sent to another work that will try to localize those errors.

Assigning top 5 cited papers from this thesis according to the specified criteria

To demonstrate how the N-dimensional approach can be used, the top 5 cited papers used in this thesis have been selected and classified in Table 2.2.

Paper	Towards Highly Reliable Enterprise Network Services [10]	Failure Diagnosis Using Decision Trees [21]	SherLog: Error Diagnosis by Connecting Clues from Run-time Logs [138]	Event-Tree Analysis Using Binary Decision Diagrams [5]	NEVERMIND, the Problem Is Already Fixed [52]
Data sources	packets, traceroute measurements, configuration files	log files	log files, source code	(paper do not focus on this)	resource utilization
Implementation	probabilistic inference graph	deterministic, decision trees	deterministic, inference model	deterministic, decision tree	deterministic, inference model
Affecting monitored network	yes, hybrid	no	no	no	no
Knowledge base	shallow	shallow	shallow	deep	shallow
Extendible	no	no	yes	yes	yes
Using training data	yes	yes	no	no	yes
Automation	fully automatic	semi, user needs to label input data	fully automatic or semi automatic	fully automatic	fully automatic
Location of execution	central server	central server	end devices	(paper do not focus on this)	central server
Error types	availability and performance	availability and application	application	any	performance
Use-cases	single, enterprise networks	single, internet sites	single, software	multiple, universal approach	single, DSL lines
Cooperation	no	no	no	no	no
Time of diagnostics	reactive	reactive	reactive	reactive	proactive
Diagnostic steps	detection and localization	localization	localization	detection and localization	detection and localization

Table 2.2: Classification of top 5 cited papers used in this thesis according to the N-dimensinal approach.

Chapter 3

Research Summary

This chapter summarizes the research contribution of the thesis. Six different methods for network error detection are presented. While each method focuses on detecting applications and network errors by analyzing network communication, they differ by theory applied, complexity, and outcomes. In a system-wide view, these methods can be considered complementary. However, they overlap in the set of possible diagnosed types of errors, which can be used to collaborate evidence found and provide richer information about the error detected. Figure 3.1 shows all methods described in this work, that are grouped based on the approach.

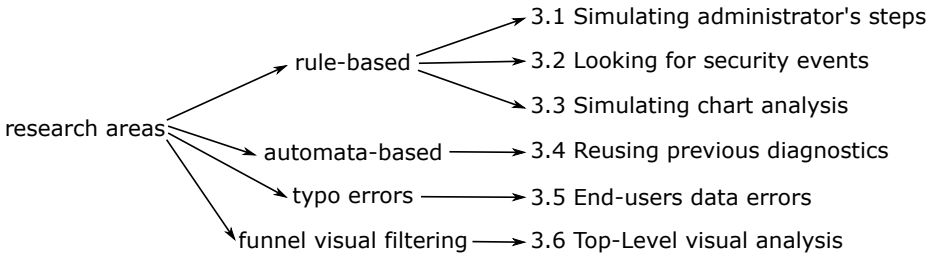

Figure 3.1: Research areas and diagnostic methods on which this dissertation thesis focuses. Numbers in front of methods represent the numbering of sections.

The first three methods are rule-based, requiring one to specify or infer these rules in advance. While this requires a substantial manual effort to define and refine the rules, the advantage is the explainability of this approach. Providing a comprehensive description of the problem found is valuable for network operators. Moreover, in many cases, the description may also contain evidence, particularly a trace of network communication for further analysis or reproduction of the problem.

The fourth method can learn automatically from examples to construct expected and erroneous communication models. The method employs finite automata to describe correct and wrong communication traces from provided communication samples. The long process of manual rule definition is replaced by automatic model construction. The experiments suggest that only a few samples are required for a reliable model.

The fifth method uses a dictionary with all the identifiers (such as email addresses) used inside the monitored network. Based on this dictionary, the method can recognize a typographical error (or just typo) that the network's end-users have made. In addition to just detecting a typo error, the method also proposes a correction candidate. The pro-

posed algorithm uses a heuristic to score possible candidates based on probability. The evaluation shows that this approach is helpful for identifiers from the application layer.

Despite the immense efforts in workflow automation, most of the work still relies on manual data exploration and analytical insights by domain specialists. The last, sixth method describes the tool that supports the analytical work of network and security operators. We present the filtering funnel metaphor for exploring packet capture (PCAP) files by visualizing linked filter steps. We have created PCAPFunnel, a novel tool that improves the user experience and speeds up packet capture data analysis. The tool provides an overview of communication, intuitive data filtering, and details of individual network nodes and connections between them.

The following sections describe all the new proposed diagnostic methods. All the methods were implemented as software prototypes. Several experiments with artificial traffic and real captured traffic have been performed. Using the created prototypes makes it possible to identify several non-trivial errors. In addition to software prototypes, the first method was implemented inside the commercial network monitoring solution.

3.1 Rule-based Diagnostic Decision Tree

The first method was designed to create an easily maintainable database of network problems that can be used to simulate real network administrators' actions. The database must be specified by rules that are easily understandable by administrators and at the same time, the set of rules can be extended to cover new errors. Part of simulating real administrators is the requirement that the outputs from such a system can be easily trackable.

The decision tree algorithm with a newly developed form of rules has been chosen. By using the decision trees, it is possible to have *if-then-else* rules that can be understood even by people without deep programming skills. The disadvantage is that only predefined errors can be diagnosed. The decision tree can produce inaccurate conclusions when there is a new and unknown error. The principles of the method and overview of the system built from these principles are described in this section.

Using decision trees to detect problems is done by evaluating the predefined conditions for actual network data until reaching the tree leaf that contains the error, or there are no more tree nodes to continue with the diagnostic process. The decision tree imitates the diagnostic process that is done manually by internally answering simple questions, as shown in Figure 3.2. An administrator tries to find answers to these questions in a predefined order based on the previous results. In addition to a final conclusion, the tree can also produce diagnostic output during the tree traversal.

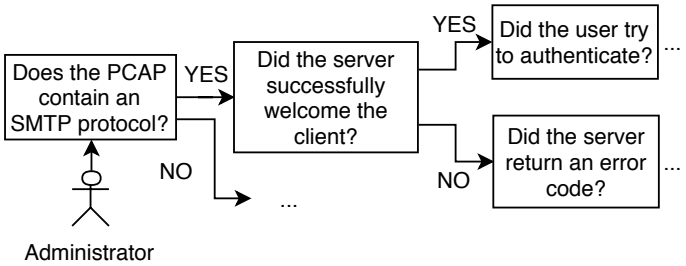

Figure 3.2: A simple illustration of a binary decision tree. The administrator diagnoses a problem by checking questions in the predefined order.

The decision tree consists of the following parts:

1. **branch nodes** describing the search for specific events in the data (for example, user authentication to the server),
2. **leaf nodes** describing the root causes of individual problems (for example, incorrect name or password),
3. **a system** that connects individual nodes and, based on whether the specific events were found, continues to pass through the tree.

Each tree node is associated with a rule with a configuration that is easily understandable even by administrators without programming skills. If an administrator wants to cover new problems by a tree, it is only necessary to create appropriate new rules and link them to existing ones. To make the decision tree more understandable, the tree is split into several subtrees executed independently. Each subtree represents one protocol for which the tree contains all the necessary rules. Table 3.1 shows for which protocols the decision trees were implemented. The table also contains the complexity of the trees represented by the number of tree nodes and detectable events. A detectable event is a particular situation that has been detected that can help an administrator with diagnostics.

Protocol	**Tree nodes**	**Fact finders**	**Events**		
			Success	**Warning**	**Error**
DHCP	24	22	10	9	4
DNS	12	12	8	4	5
FTP	24	10	15	6	7
HTTP	3	3	2	1	1
ICMP	4	2	0	0	4
IMAP	15	8	7	3	9
POP	21	7	5	10	7
SIP	38	22	15	1	8
SLAAC	8	7	1	6	1
SMB	27	25	20	3	5
SMTP	17	13	9	6	9
SSL	2	2	2	0	1
TCP	10	10	0	7	2

Table 3.1: Supported protocols and amount of rules and success, warning, and error events that describe various protocol behavior situations.

As can be seen from Table 3.1, an event can represent a successful, warning, or error situation. An example of a successful event is when a client creates a connection with a server. A warning event can describe a situation that does not need to be problematic, but the administrator should check it anyway. For example, after the client established the connection with the server, the client skipped the authentication part and started transferring the data from the server. Moreover, an error event can represent the situation when the server refuses to send some data to the client.

When applying the created method to real data, it turned out that the processing takes a very long time for larger and more complicated inputs and is necessary to optimize this

process. This problem was focused on in the paper *Using Rule-Based Decision Trees for Automatic Passive Diagnostics of the Network Problems* by creating indexes from input data and storing them in a separate database. This solution does not affect further data processing and can be applied to any diagnostic method based on data from TShark.

Another advantage of using an indexed data database is that it allows other data sources to be involved in the diagnostic process. Specifically, the paper describes the integration of log file analysis. All log data suitable for diagnostics were indexed and stored in the database, like data from TShark. Within the decision tree, which was used as a model describing the diagnostic process, it is possible to make decisions based on data from log files in the same way as from captured network traffic. However, the article did not cover the deep analysis of diagnostics from additional data sources because the primary goal of this dissertation is diagnostics based on captured network traffic.

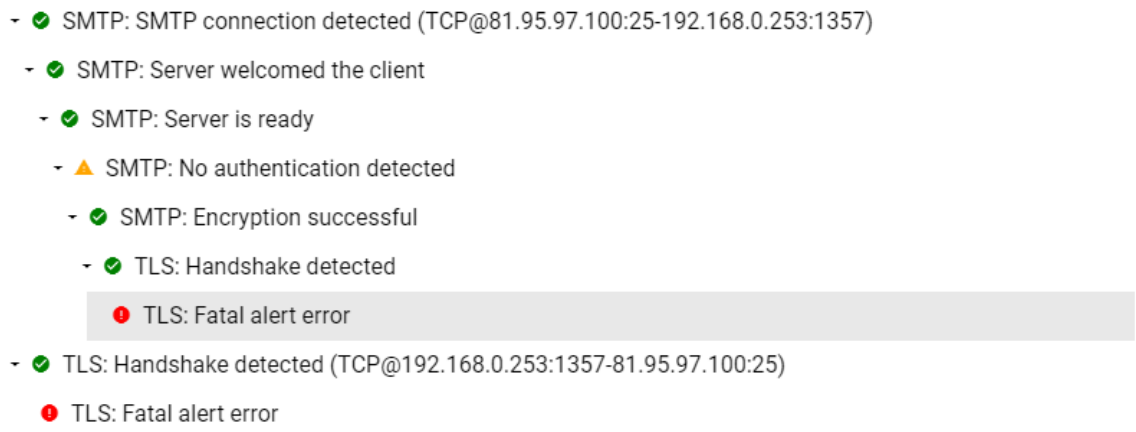

Figure 3.3: The figure shows a screenshot from the Flowmon Packet Investigator that contains a list of detected events.

Even if the rule-based approach may look outdated these days when the majority of researchers focus on machine-learning approaches, it was demonstrated that this approach is usable in real-world scenarios. The tool was integrated into a business product, Flowmon Packet Investigator, which focuses on diagnosing errors from captured network packets. The screenshot from the actual product can be seen in Figure 3.3 and Figure 3.4. This idea with results was also published in two papers:

- Holkovič Martin a Ryšavý Ondřej. *Network Diagnostics Using Passive Network Monitoring and Packet Analysis.* In: ICNS 2019. The Fifteenth International Conference on Networking and Services. Athens: The International Academy, Research and Industry Association, 2019, s. 47-51. ISBN 978-1-61208-711-5.
- Holkovič Martin a Ryšavý Ondřej. *Using Rule-Based Decision Trees for Automatic Passive Diagnostics of the Network Problems.* International Journal on Advances in Networks and Services, roč. 2020, č. 1, s. 1-10. ISSN 1942-2644.

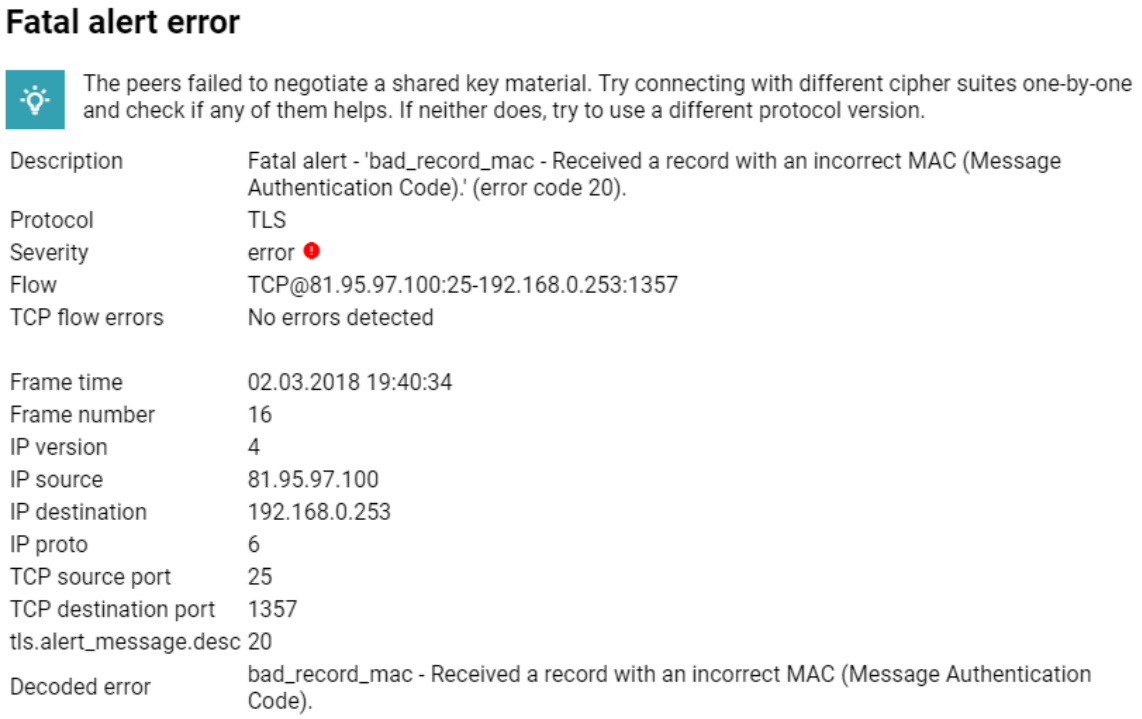

Figure 3.4: The screenshot from the Flowmon Packet Investigator that shows a suggestion for a detected event.

3.2 Security Analysis Based on Rule-based Packets Grouping and Searching

The second research domain was centered on diagnosing errors by applying security analysis. Sometimes, service failure can be linked to a present attack on the network, and by identifying the attack, it becomes possible to uncover the root cause of the issue. Currently, solutions such as firewalls or IDS can reliably detect most attacks. However, these systems operate on a per-flow or per-packet basis due to the detection speed. This subsection presents a method that can detect attacks despite different flows without restrictions on selected packets or protocols.

The principle of the detection algorithm is shown in Figure 3.5. The system divides packets into several disjunctive groups according to any packet attribute. Within each group, packets with specific attributes are then searched. Various operations and conditions are then applied to the found packets, which must be met in order for the searched event to be marked as seen. Each group that fulfills all assert conditions increases a counter. If the counter reaches a predefined value, the event the system looks for is detected.

The created method can search for events across multiple flows, making the search process more flexible. Because the search is not limited to a per-flow or single-packet analysis, the processing is more complex, and the speed is significantly lower than existing IDS solutions. Therefore, the proposed method should not replace the existing solutions but should be used in addition to existing IDS solutions. Another benefit of the created

method is that an administrator can use a user-friendly rule language to extend the list of events the system seeks.

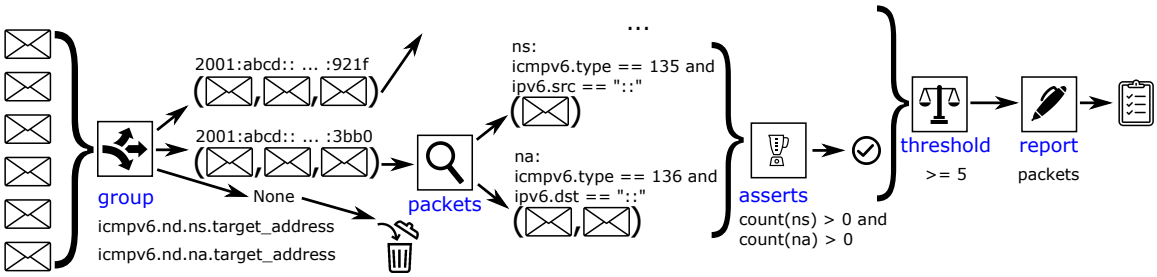

Figure 3.5: The main idea of detecting security events is based on the flexible way of grouping packets into groups and searching inside those groups.

For demonstration purposes and to evaluate the method, we have created 35 rules for different kinds of events, e.g., MitM ARP attack, HSRP protocol configuration with nonoptimal configuration, network scanning, using old-unsecured TLS version. The method's functionality for the created tools demonstrates that the method can properly detect events that may be related to a diagnostic process. The output of the tool is shown in Figure 3.6. It can be seen that part of the detected event is a list of packets that contributed to the event detection. More information about the proposed method can be found in the paper:

- Holkovič Martin, Ryšavý Ondřej a Dudek Jindřich. *Automating Network Security Analysis at Packet-level by using Rule-based Engine.* In: Proceedings of the Sixth European Conference on the Engineering of Computer-Based Systems. Bucharest: Association for Computing Machinery, 2019, s. 1-8. ISBN 978-1-4503-7636-5.

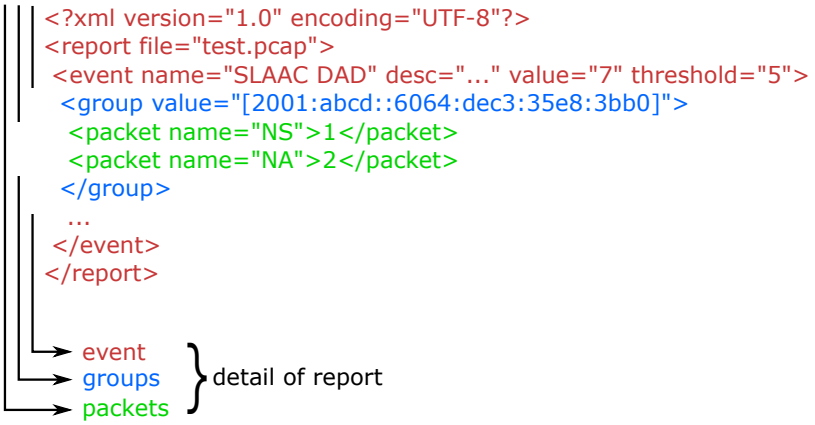

Figure 3.6: The output from the tool that has implemented the proposed method.

3.3 Using Pattern-Based Analysis for Diagnosis

The next research area is similar to the first one in a way that the created method tries to simulate the manual analysis of a real administrator. However, in this case, the method analyzes data usually displayed in charts, and administrators look for specific patterns

inside those data. An example of such a pattern can be a peak in a data series or a crossing of the lines from multiple data series.

We have presented a new rule-based approach to detecting and identifying network issues. The rules employ patterns that consist of a sequence of value changes to identify a sequence in network communication that represents an anomaly. This new approach automates the labor activity conducted by network administrators that use the visual representation of network activities to identify non-standard situations.

The main idea of this approach can be seen in Figure 3.7, where an error with the DNS server is indicated by a decreased amount of correct replies and an increased amount of replies with an error. An administrator can diagnose the problem more easily and quickly by detecting such an anomaly inside the network.

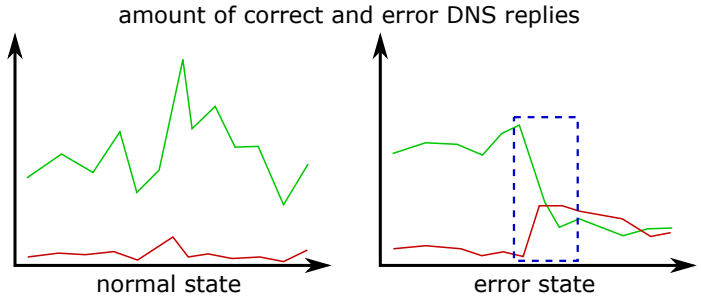

Figure 3.7: Visualization of DNS replies during the normal and error state. The green line represents the number of normal replies and the red line replies with an error.

The proposed method is not using any image processing algorithm. The pattern search system uses simple descriptions of value changes, which are easily understandable by network administrators. An example of a pattern is a rapid drop followed by a sharp increase, which can be seen as a V-shape in the traffic graph. Administrators mostly use this form of visual analysis to get an overview of network status or to observe specific host behavior.

The process of pattern lookup starts with assigning all the packets into time intervals based on the arrival time. In the next step, the system will apply the aggregation function on multiple packets within the same time interval. An example of such a function is the counting of replies with an error code. An alphabet character abstracts the numbers that are output from the aggregation functions according to the previous and current values. After concatenating all the characters together, a string value is created that represents the chart data. The pattern lookup process looks for patterns by evaluating regular expressions over those strings. This process is visualized in Figure 3.8, in which data is represented by a string with arrows, where one arrow is one data point.

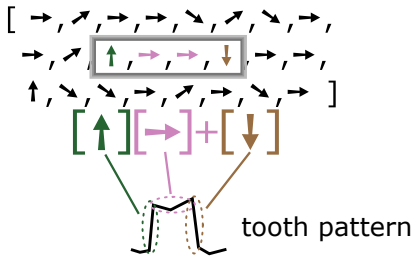

Figure 3.8: The process of pattern lookup is implemented as a search by regular expressions.

To demonstrate the tool's functionality, we have tested the tool over a small amount of network data. One such output can be seen in Figure 3.9. Even though the tool can detect complex situations, the figure shows only a very simple example to illustrate the idea. The diagnostic method has detected the drop jump pattern, which means that the QoS queuing and buffering settings are applied to the application. In the case of multimedia applications, these settings can negatively affect the user's quality of experience.

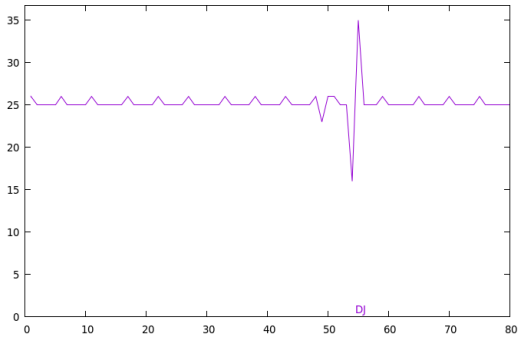

Figure 3.9: Analysis of the transfer speed of the selected application. The chart shows the number of transferred packets in 20 ms time intervals.

The proposed method was published in the paper:

- Holkovič Martin, Bohuš Michal a Ryšavý Ondřej. *Pattern Detection Based Network Diagnostics*. In: Proceedings of the 17th International Joint Conference on e-Business and Telecommunications. Setubal: SciTePress - Science and Technology Publications, 2020, s. 35-42. ISBN 978-989-758-445-9.

3.4 Creating Automata Models for Diagnostics of Repetitive Problems

Computer network errors can be repetitive in nature, occurring periodically or consistently over time. These repetitive errors can have various causes, such as hardware or software issues, configuration problems, or network congestion. The next research effort was focusing on these errors. The research goal was to learn from previous diagnostic results and reuse the learned knowledge in the future. Suppose an administrator finds and diagnoses a network error (manually or using another method). In that case, saving the error traffic and the diagnostic description is possible.

The proposed approach creates one model per application protocol valid only for the networks that produce similar traces as traces in the training data. When providing generic traces, the model will also be generic and, therefore, usable for more networks. On the other hand, if specific training data are provided that capture the uniqueness of the network configuration, the created model will work only for that specific configuration. With these specific models, it is possible to detect extraordinary situations that could be problematic in one network, even if the same situation can be expected in another network.

A finite-state automaton model was used to learn from the previous diagnostics. The model describe all the possible behaviors of the selected protocol. Pairs of request-reply messages represent the behavior and are saved as the automaton states and edges. One pass of the automaton represents one communication consisting of requests and paired replies to

them. For this reason, the created method is suitable only for application protocols using the request-reply communication pattern. An example of a protocol model is in Figure 3.10.

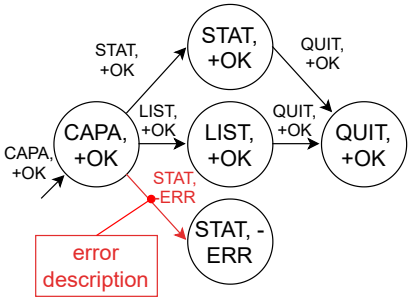

Figure 3.10: An automaton model that describes the small part of the POP protocol model. Labels above the edges correspond to the "request, reply" values.

Creating a model consists of two steps:

1. The administrator will provide a list of PCAP files containing error-free communication of the protocol. The aim is to provide as much variability of these communications as possible so that the model covers even less frequent situations. The result is a model that describes the correct behavior of the selected protocol.

2. In the second step, the administrator extends the created model with new communications, but this time it needs to be fixed, for which the administrator needs to provide an adequate description. The result is a model extended by erroneous transitions to which individual descriptions of errors are noted.

During diagnostics, the model may find itself in a state where it does not know how to proceed because it lacks a transition. In this case, the diagnostics end with an unknown error. Such a result informs the administrator that communication does not correspond to the expected model.

The research on this topic has been split into two parts. The first part uses the automata model described in the previous text. However, based on the experience with the model, the model had made changes to the form of timed automata. That was the focus of the second part. In addition to a "request-reply" value, the model based on timed automata also contains a time delta that represents seconds since the previous "request-reply" was received. With transitions recognizing the time information, it is possible to cover more situations. For example, it is possible to catch performance issues. An example of a timed automata model is shown in Figure 3.11.

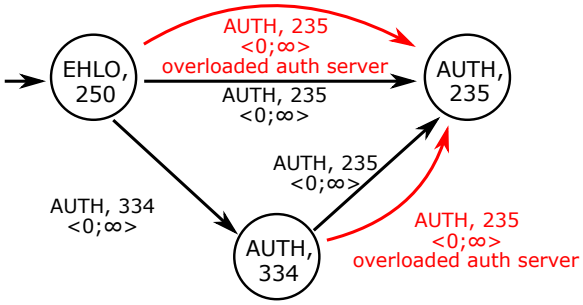

Figure 3.11: A subset from the model describing the SMTP finite automata model.

The method was tested on four protocols of various complexity: DNS, SMTP, POP, and FTP. The results from testing are shown in Table 3.2. The accuracy depends on the protocol and it ranges from 63% to 100%. All situations in which the method incorrectly diagnosed the analyzed errors were caused by the fact that the model was not sufficiently trained from enough unique input communications. When deployed in a real network, it can be expected that the model will be incomplete from the beginning and require learning new situations. Although the model does not cover all possible situations, it helps diagnose repetitive errors. As the model can learn errors during deployment, an administrator must not deal with the same errors again.

Protocol	**Testing scenarios**		**Testing results**				
	Successful	**Failed**	**TN**	**TP**	**FN**	**FP**	**Accuracy**
DNS	6	2	4	1	1	2	63 %
SMTP	2	1	2	1	0	0	100 %
POP	6	2	6	2	0	0	100 %
FTP	18	6	18	5	1	0	96 %

Table 3.2: The correct results are shown in the true negative (TN) and true positive (TP) columns. The columns false positive (FP) and false negative (FN) on the other side contain the number of wrong test results. The ratio of correct results is calculated as a true/false ratio (Accuracy).

The proposed method can save time for experienced administrators because they can create a model containing frequent errors on the network that can be automatically checked when diagnosing a new problem. One benefit is that having a trained model will represent the protocol in a configuration specific to the monitored network and will not accept situations that may be valid only for other networks. Because the trained models can be easily shared, the method allows less experienced administrators to use models created by experienced administrators. The proposed method was published in two papers:

- Holkovič Martin, Ryšavý Ondřej and Polčák Libor. *Using Network Traces to Generate Models for Automatic Network Application Protocols Diagnostics*. In: Proceedings of the 16th International Joint Conference on e-Business and Telecommunications Volume 1: DCNET, ICE-B, OPTICS, SIGMAP and WINSYS. Praha: SciTePress - Science and Technology Publications, 2019, pp. 37-47. ISBN 978-989-758-378-0.
- Holkovič Martin, Polčák Libor and Ryšavý Ondřej. *Application Error Detection in Networks by Protocol Behavior Model*. In: Communications in Computer and Information Science. Praha: Springer Verlag, 2020, pp. 3-28. ISBN 978-3-030-52685-6.

3.5 Correcting User's Data According to Typographical Errors Analysis

The possibility of diagnosing network issues by analyzing end-user input stands for complementary approaches to previously presented techniques. For example, when configuring an email client application on a smartphone, an end-user must specify several parameters, such as server name, port number, or login name. The email application would not work correctly if any of these parameters were misconfigured. This research area aims to detect these types of errors (we call them typo errors) and report them to an administrator.

Score	Wrong word	Operation
25.0	GOOGLE.COM	Capitalization
23.9	g0ogle.com	Substitution for a similar symbol
23.6	ggogle.com	Substitution with a duplicated symbol
23.6	giogle.com	Substitution in a fat finger distance
23.3	gwogle.com	Substitution
22.0	gogle.com	Deletion of the repeated symbol
21.8	gooogle.com	Insertion of a repeated symbol
21.5	goole.com	Deletion
21.2	giigle.com	Substitution in a fat finger distance
21.2	google.comp	Insertion at the end of the word
20.6	gooqgle.com	Insertion
19.7	ogogle.com	Transposition
18.5	gopople.com	Insertion in a fat finger distance
17.0	ggle.com	Deletion
15.2	goqoqgle.com	Insertion

Table 3.3: A list of misspelled words originated from the word google.com and their score representing the probability of errors that caused the misspelled word.

Before detecting typo errors, it was necessary to determine in which data the created algorithm would be looking for errors. Since this is an analysis of typographical errors, it does not make sense to analyze data that is not entered directly by the user but is created by the machine. This data can be divided into application data (according to the TCP/IP model) and lower-layer data. Examples of application data are a login name, email address, or domain, and for lower layer data, it is an IP address or transport port. Lower layer data can also be entered by the user, for example, when configuring some applications.

In addition to detecting typo errors in user data, the method should estimate which word the user meant. During the diagnostics, the administrator can provide the user with the exact description of solving the problem. A heuristic was created to estimate the word that the user thought. The basis of this heuristics is the minimum edit distance, but it also tries to take into account the most common types of errors that can occur. An example of such an error is pressing the wrong key on a keyboard (also called a fat finger) or typing a letter with a similar phonetic. The most significant difference of this method compared to classical spellcheckers is that the analyzed words are not based on any language grammar. At the same time, using the context from the surrounding text is not possible.

Table 3.3 shows some misspelled words caused by misspelling the word „google.com“. For all these misspelled words, a list of operations was determined that had to be applied in order for the correct word to become a given misspelled word. Subsequently, according to this list of operations, a score is calculated that considers the probability of the operations. The table is sorted from the word with the highest (probability) score to the lowest.

The main idea of this diagnostic approach is shown in Figure 3.12. In the first part, the tool detects misspelled words in the input data, creating a set of candidates of possible correct words. Each candidate is then evaluated with the created heuristics. Of all the candidates, the one with the highest score (probability) is selected and reported to an administrator.

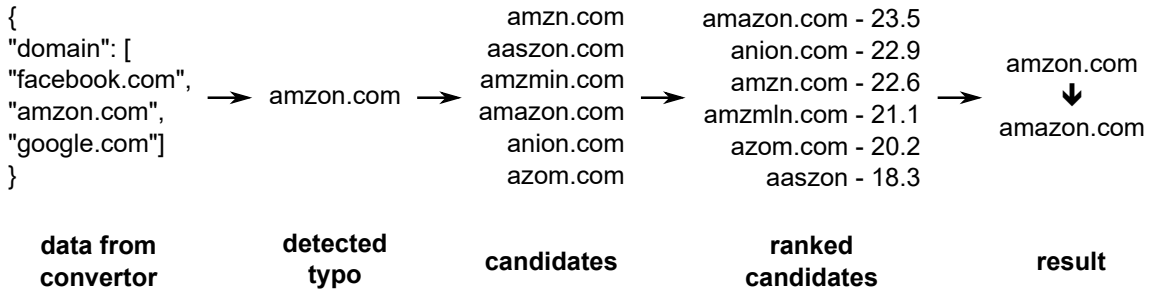

Figure 3.12: The principle of the diagnostic method based on detecting the user typo errors.

One basic problem is determining which words are correct and which contain a typo. A dictionary of words is used for this, while only the words in the dictionary are considered correct. The same dictionary is also used to create candidates to suggest the correct word. This research was not focused on analyzing how dictionaries containing the correct words can be created (or filled). One of the possible assumptions is that dictionaries can be created manually (for example, by exporting all email addresses used within the company) or by using an external tool that can learn the correct words on its own. One of the possible tools for automatic learning is the tool described in section 3.4, which uses finite automata to distinguish correct communication from erroneous one. If the communication is correct, it is possible to mark all used words as correct.

To test and evaluate the proposed method, we have applied most common typo errors to the most visited websites and usernames used in our faculty. Most of the errors were correctly fixed, for example domain "chasr.com" was fixed to "chase.com". However, a few errors were fixed to wrong values, for example username "ikuceran" was fixed to "ikuceraj" instead of "ikucura". Table 3.4 shows the accuracy of corrections.

Data type	domains	usernames
Word count	1000	100
Detected as error	991	100
Detected as correct word	9	0
Correct best candidate	967	99
Multiple best candidates	4	0
Wrongly repaired	20	1
Success rate	97.5% (967 of 991)	99% (99 of 100)

Table 3.4: Results from testing the correction of typo errors in domain names and usernames.

As can be seen from Table 3.4, the tool found errors with high accuracy in the words from the application TCP/IP layer (domains and login names). Specifically, the accuracy was 96.5% and 99%. However, when we tried to evaluate the methodology for words from the lower layers (IP addresses, transport ports), the accuracy was very low. The reason is that the words from application layers are optimized for use by end-users and are easily distinguishable, which makes it easier to apply typo errors correction techniques. Conversely, the words from the lower layers are optimized for the best possible computer processing and many times it is hard to fix, or even detect, the typo errors. For example,

login names within our faculty are words like "vecerav", "polcak" or "iletavay". On the other hand, IP addresses inside of our faculty were "147.229.176.14", "147.229.176.19," or "147.229.176.8". Therefore, the evaluation showed that the developed methodology is only suitable for some data types. More detailed information can be found in the following paper:

- Holkovič Martin, Bohuš Michal a Ryšavý Ondřej. *Network Problem Diagnostics using Typographic Error Correction.* 17th International Conference on Network and Service Management (CNSM). IEEE, 2021. ISBN 978-3-903176-36-2.

3.6 Top-Level Visual Analysis of Network Traffic

All the previous works have been focused on automatic computer errors diagnostics. However, during the process of experimenting with new methods of diagnostics on real networks, we have found out that even if some methodology detects some problem, it usually doesn't provide enough information for network administrators. For each detection, an administrator wants to know whether the detection is false positive, what's the impact on the whole network and many other details that can not be easily extracted from the input file automatically.

The usual diagnostic pipeline consists of three steps executed in this order:

1. Executing **automatic tools** that can automatically detect and ideally also find the root cause of network problems.

2. After a problem is detected, the administrator performs a **manual top-level analysis** over a larger amount of (usually aggregated) traffic. This analysis can bring more light into the problem, for example, what's the impact on the whole network.

3. If the previous two steps do not provide sufficient information, administrators continue with **manual in-depth traffic inspections** . In-depth analysis of network traffic is allowing the administrators to explore the detailed information and content of individual packets. Because it is very time consuming, administrators try to avoid this analysis.

This part of my research was focused on the second step of the pipeline - top-level analysis. Problem with the current analysis tools is that they have a steep learning curve and only limited visualization capabilities. I have together with other co-authors discussed several experienced network administrators (from both business and academia) and we have identified five common tasks that the tool needs to easily handle:

1. identify the top N communication sources, based on given criteria,

2. discover unusual patterns in the network traffic (e.g., peaks),

3. identify nodes with which the particular station communicated,

4. identify nodes providing specific services to a network (e.g., DNS server),

5. share the analysis parameters with coworkers.

Our new approach is based on the filtering funnel metaphor to support filtering and data analysis based on linking several independent filter steps. The filtering funnel metaphor is based on several layers of filters, where output from one layer is an input of another layer. Applying several filters in this metaphor reduces the amount of data in each layer. The amount of data then looks similar to a funnel. All the filter steps are permanently visible to the user, who can interactively modify filters' parameters. With this new approach, a domain expert can quickly check the data's content, determine its structure, analyze network actors' behavior, or reveal the cause of network issues. The user interface leverages linked views and conventional visualizations, so it is also suitable for novice users. The main idea of our funneling metaphor is displayed in Figure 3.13.

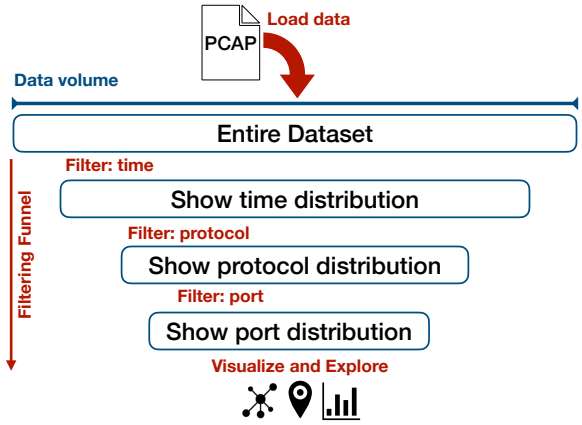

Figure 3.13: The filtering funnel metaphor illustration.

We have created a web implementation that takes captured network traffic in PCAP file format. However, we think that the same approach can be applied to other data sources as well, such as NetFlow records or log files. The tool allows the administrator to either create any filter layers in any order or just to use one of the predefined filters combinations. Another very useful feature is that when the analysis is finished, it is possible to export all the used settings and use them later or just share them with another administrator.

Each analysis always starts by visualizing the amount of transferred traffic over time. In this visualization, it is possible to see some unexpected patterns, outliers or anomalies in general. The tool's user can select only the subinterval of the data that will be further analysed. Example of this visualization is shown in Figure 3.14.

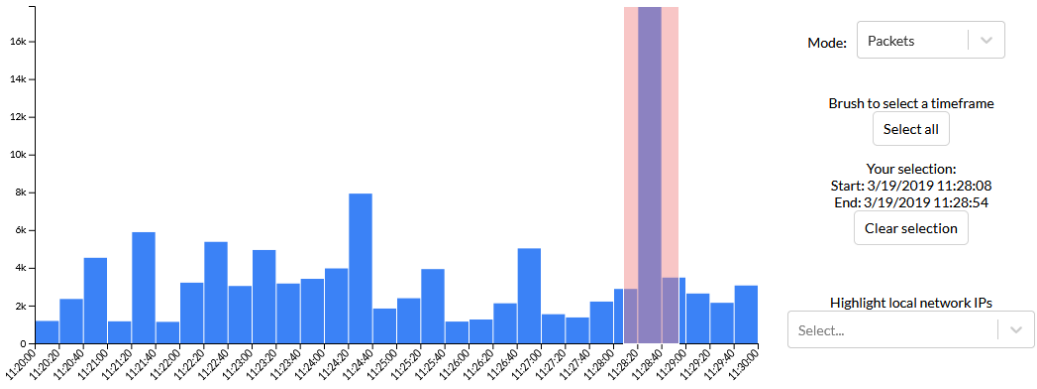

Figure 3.14: Filtering the data based on the time interval.

After selecting the interval that should be analyzed, several independent filtration layers are used. Each layer can filter the data according to IP addresses, network protocols, transport port numbers, transport protocols, or application names. The layer consists of two parts, a table and a mini chart. The table is showing the connections order based on the filtration criteria. In the case of filtering according to the IP addresses, the table contains IP addresses ordered according to the amount of transferred traffic. Transferred traffic can be calculated in bytes, packets, connections or two way connections. In the table it is possible to select rows that should be analyzed further. Also the selected rows are visualized in the second part of this layer, the mini chart. Similarly to the previous step, the chart shows the amount of transferred data in time. However in this case, the colors differentiate the selected rows from the rest of the traffic. Example of a filtration layer is shown in Figure 3.15.

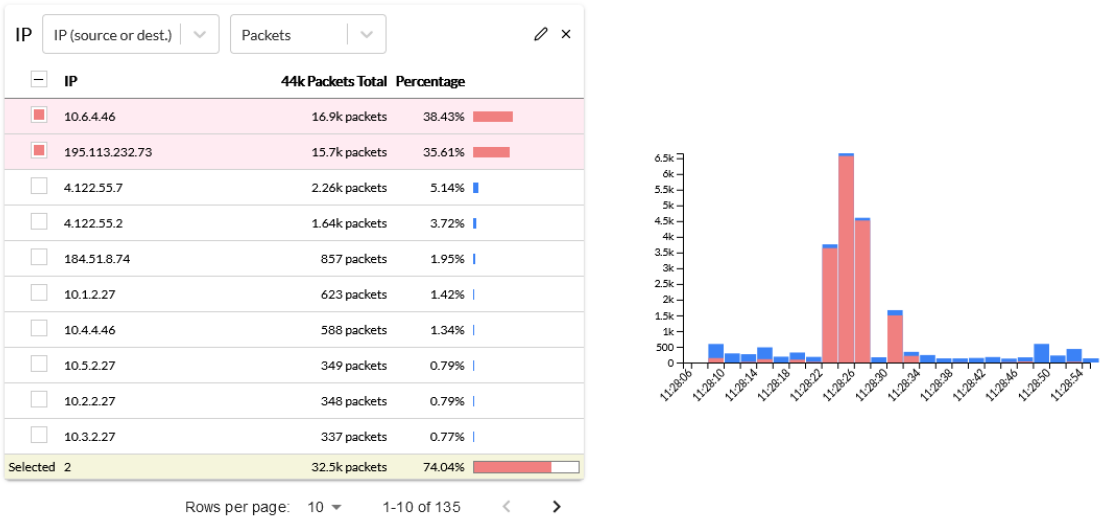

Figure 3.15: Example of a filtering layer that filters the data based on the selected IP addresses.

When the data passes through all the filtration layers, the network graph is shown. Inside the network graph, it is possible to click on any node which represents one communication entity or on any link between two communication entities. After the click an additional subpage is shown that consists of communication profile, network profile, packets property statistics and even raw packets. The preview of a page with all these sections is shown in Figure 3.16.

Figure 3.16: Detailed view on the network node.

This tool is different from the previous tools described in this chapter, that the tool's goal is not to automatically detect network errors but this tool is aimed to support administrators during their manual analysis. Because of this difference, the evaluation is also different from the previous tools. We have contacted nine domain experts and asked them to solve five different tasks with provided PCAP files. Their goal was to find answers for several questions related to network or security network operations. Overall, the participants engaged well with the tasks and their feedback was mostly positive.

Because this method does not contain any built in knowledge, the evaluation was done differently. Instead of testing the created prototype on selected errors, the evaluation was done by engaging real network administrators and asking them for feedback. We were able to connect with nine domain experts from both academia and the private sector. After we gave them all the instructions and short tutorial, they needed to use the tool to perform five tasks, such as identifying the application with the most significant number of connections or identifying nodes providing DNS server capability on non standard ports. We ask the domain experts to fill a post-study questionnaire to rate the usability of the tool. All answers can be seen in Figure 3.17 however, the result in the short term is that the tool, and the methodology, is very flexible and supportive during the initial packet capture analysis.

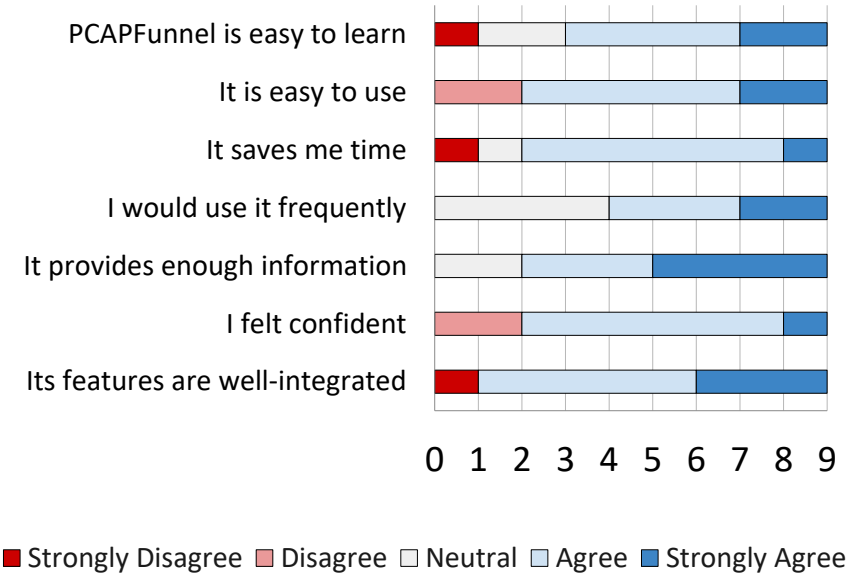

Figure 3.17: Answers from the domain experts from the post-study questionnaire regarding the usability of the tool.

More information about the research topic can be found in the published paper:

- Uhlár Juraj, Holkovič Martin a Rusňák Vít. *PCAPFunnel: A Tool for Rapid Exploration of Packet Capture Files.* In 2021 25th International Conference Information Visualisation (IV). To appear. United States of America: The Institute of Electrical and Electronics Engineers, Inc., 2021. s. 69-76. ISBN 978-1-6654-3827-8.

3.7 List of Outcomes

This section contains a list of all my publications - both included (Subsection 3.7.1) and not included in this thesis (Subsection 3.7.2), a list of research projects in which I was a contributing member (Subsection 3.7.3), a list of created software (Subsection 3.7.4), and a list of students theses I have supervised (Subsection 3.7.5). The full versions of the included papers are located in Appendix A.

3.7.1 Papers Included in Thesis

1. Holkovič Martin and Ryšavý Ondřej. **Network Diagnostics Using Passive Network Monitoring and Packet Analysis**. In: ICNS 2019. The Fifteenth International Conference on Networking and Services. Athens: The International Academy, Research and Industry Association, 2019, pp. 47-51. ISBN 978-1-61208-711-5.
   Author's participation: 80%

2. Holkovič Martin, Ryšavý Ondřej and Polčák Libor. **Using Network Tra-ces to Generate Models for Automatic Network Application Protocols Diagnostics**. In: Proceedings of the 16th International Joint Conference on e-Business and Telecommunications Volume 1: DCNET, ICE-B, OPTICS, SIGMAP and WINSYS. Praha: SciTePress - Science and Technology Publications, 2019, pp. 37-47. ISBN 978-989-758-378-0.
   Author's participation: 70%

3. Holkovič Martin, Ryšavý Ondřej and Dudek Jindřich. **Automating Network Security Analysis at Packet-level by using Rule-based Engine**. In: Proceedings of the Sixth European Conference on the Engineering of Computer-Based Systems. Bucharest: Association for Computing Machinery, 2019, pp. 1-8. ISBN 978-1-4503-7636-5.
   Author's participation: 85%

4. Holkovič Martin, Bohuš Michal and Ryšavý Ondřej. **Pattern Detection Based Network Diagnostics**. In: Proceedings of the 17th International Joint Conference on e-Business and Telecommunications. Setubal: SciTePress - Science and Technology Publications, 2020, pp. 35-42. ISBN 978-989-758-445-9.
   Author's participation: 50%

5. Holkovič Martin and Ryšavý Ondřej. **Using Rule-Based Decision Trees for Automatic Passive Diagnostics of the Network Problems**. International Journal on Advances in Networks and Services, vol. 2020, no. 1, pp. 1-10. ISSN 1942-2644.
   Author's participation: 90%

6. Holkovič Martin, Polčák Libor and Ryšavý Ondřej. **Application Error Detection in Networks by Protocol Behavior Model**. In: Communications in Computer and Information Science. Praha: Springer Verlag, 2020, pp. 3-28. ISBN 978-3-030-52685-6. ISSN 1865-0929.
   Author's participation: 70%

7. Holkovič Martin, Bohuš Michal and Ryšavý Ondřej. **Network Problem Diagnostics using Typographic Error Correction**. In: Proceedings of the 17th International Conference on Network Service Management (CNSM 2021). Izmir: Institute

of Electrical and Electronics Engineers, 2021, pp. 482-490. ISBN 978-3-903176-36-2.
Author's participation: 60%

8. Uhlár Juraj, Holkovič Martin and Rusňák Vít. PCAPFunnel: **A Tool for Rapid Exploration of Packet Capture Files**. In: 2021 25th International Conference Information Visualisation (IV). Sydney: IEEE Biometric Council, 2021, pp. 69-76. ISBN 978-1-6654-3827-8.
Author's participation: 30%

3.7.2 Other Relevant Papers

1. Hranický Radek, Holkovič Martin, Matoušek Petr and Ryšavý Ondřej. **On Efficiency of Distributed Password Recovery**. The Journal of Digital Forensics, Security and Law, vol. 11, no. 2, 2016, pp. 79-95. ISSN 1558-7215.
Author's participation: 30%

2. Polčák Libor, Caldarola Leo, Cuda Davide, Dondero Marco, Ficara Domenico, Franková Barbora, Holkovič Martin, Choukir Amine, Muccifora Roberto and Trifilo Antonio. **High Level Policies in SDN**. In: Communications in Computer and Information Science. Berlin: Springer International Publishing, 2016, pp. 39-57. ISBN 978-3-319-30221-8. ISSN 1865-0929.
Author's participation: 7%

3. Polčák Libor, Holkovič Martin and Matoušek Petr. **Host Identity Detection in IPv6 Networks**. In: E-Business and Telecommunications. Berlin: Springer Verlag, 2014, pp. 74-89. ISBN 978-3-662-44787-1. ISSN 1865-0929.
Author's participation: 20%

4. Polčák Libor, Holkovič Martin and Matoušek Petr. **A New Approach for Detection of Host Identity in IPv6 Networks**. In: Proceedings of the 4th International Conference on Data Communication Networking, 10th International Conference on e-Business and 4th International Conference on Optical Communication Systems. Reykjavík: SciTePress - Science and Technology Publications, 2013, pp. 57-63. ISBN 978-989-8565-72-3.
Author's participation: 20%

5. Špaček Stanislav, Velan Petr, Holkovič Martin and Plesník Tomáš. **Event-Flow Correlation for Anomaly Detection in HTTP/3 Web Traffic**. In: 2023 IEEE/IFIP Network Operations and Management Symposium (NOMS 2023). Miami, Florida, USA: IEEE Xplore Digital Library, 2023, pp. 1-6. ISBN 978-1-6654-7717-8. ISSN 1542-1201.
Author's participation: 10%

3.7.3 Research Projects and Grants

1. **SECURIAN - Streamlining cybersecurity incident analyses**, Technology Agency of the Czech Republic, FW06010009, 2023-2025

2. **ETA - Context-based Encrypted Traffic Analysis Using Flow Data**, Technology Agency of the Czech Republic, FW03010099, 2021-2023

3. **CONCORDIA - Cyber security cOmpeteNCe fOr Research anD InnovAtion**, Horizon 2020 Framework Programme, 830927, 2019-2023

4. **BONNET - Security monitoring of ICS communication in the smart grid**, Ministry of the interior of the Czech Republic, VI20192022138, 2019-2022

5. **Smart ADS**, Technology Agency of the Czech Republic, TH04010073, 2019-2021

6. **AMANDA: Asset Management ANd DiAgnostics**, Technology Agency of the Czech Republic, TN01000077/05, 2019-2020

7. **DISTANCE - Packet analysis based network diagnostics**, Technology Agency of the Czech Republic, TH02010186, 2017-2019

8. **ICT tools, methods and technologies for smart cities**, Brno University of Technology, FIT-S-17-3964, 2017-2019

9. **Modern Tools for Detection and Mitigation of Cyber Criminality on the New Generation Internet**, Ministry of the interior of the Czech Republic, VG2010-2015022, 2010-2015

3.7.4 Software

1. Ryšavý Ondřej, Holkovič Martin, Matoušek Petr, Minařík Pavel, Aleš Šnupárek, Jan Střítežský. **A system for discovering relationships between network flows (NetFlow/IPFIX)**, Computer Software, 2022

2. Jeřábek Kamil, Minařík Pavel, Holkovič Martin. **System for detecting encrypted DNS communication**, Computer Software, 2021

3. Holkovič Martin. **Semi-automatic Network Application Protocols Diagnostics by Using Network Traces**, Computer Software, 2019

4. Januš Filip, Holkovič Martin. **Application data extractor from network protocols**, Computer Software, 2018

5. Šuhaj Peter, Holkovič Martin. **Computer networks vulnerability detector**, Computer Software, 2018

6. Nahálka Roman, Holkovič Martin. **Extractor of tunneled data into separate flows**, Computer Software, 2018

7. Dudek Jindřich, Holkovič Martin. **Networks attacks detector**, Computer Software, 2018

8. Svoboda Ondřej, Holkovič Martin. **Tool for network protocols semiautomatic diagnostics**, Computer Software, 2018

9. Hranický Radek, Holkovič Martin, Zobal Lukáš, Večeřa Vojtěch, Mikuš Dávid. **Fitcrack - a distributed password recovery tool**, Computer Software, 2016

10. Holkovič Martin. **SDN Identity Manager**, Computer Software, 2015

11. Polčák Libor, Martínek Tomáš, Hranický Radek, Bárta Stanislav, Holkovič Martin, Franková Barbora, Kramoliš Petr. **Sec6Net Identity Management System**, Computer Software, 2014

12. Polčák Libor, Martínek Tomáš, Hranický Radek, Bárta Stanislav, Holkovič Martin, Franková Barbora, Kramoliš Petr. **Sec6Net Lawful Interception System**, Computer Software, 2014

13. Holkovič Martin, Polčák Libor. **ndtrack**, Computer Software, 2013

3.7.5 Supervised Theses

1. Navrátil Petr. **Anonymization of PCAP Files**. Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2019.

2. Šuhaj Peter. **Extending NetFlow Records for Increasing Encrypted Traffic Classification Capabilities**. Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2019.

3. Bohuš Michal. **Diagnosing Errors inside Computer Networks Based on the Typo Errors**. Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2019.

4. Beňo Marek. **Phishing Detection in Web Pages**. Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2018.

5. Škápik Anton. **Detection of Volumetric DoS and DDoS Attacks in Real Time on the L3 Network Layer**. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2017.

6. Nahálka Roman. **Tunneled Data Extraction into Separate Flows**. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2017.

7. Šuhaj Peter. **Vulnerability Detection in Computer Network**. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2017.

8. Januš Filip. **Application Data Extraction from Network Protocols**. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2017.

9. Bohuš Michal. **Server Data Monitoring with Android Notification**. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2017.

10. Dudek Jindřich. **Detection of Network Attacks Using Tshark**. Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2017.

11. Svoboda Ondřej. **Network Protocols Semiautomatic Diagnostics**. Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2017.

12. Šťastný Filip. **Building Trading System Based on Time Range Breakout**. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2016.

13. Klhůfek Michal. **Use Machine Learning to Predict Future Market Prices**. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2016.

14. Gavryliuk Olga. **SDN Routing According to Transmitted Content**. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2015.

Chapter 4

Conclusions

Computer network diagnostics is a difficult task requiring a lot of knowledge and time to do it properly. Although computer network diagnostics is not a new area and there are already several available tools and published papers, it is still not a solved problem. The goal of this thesis was to create a system for the automatic diagnostics of network errors which would provide a root cause analysis even for a less experienced network administrator.

This work is a compilation thesis that has presented six newly created diagnostics methods. Each method was focused on a different type of error, and overall, the individual methods complemented each other. However, each method's basic idea was to take the input data in PCAP format, use the TShark tool to convert the data, and then look for error states in the data. Methods are not limited to a small set of network protocols, and methods should be usable by network administrators and not just researchers or programmers.

4.1 Research Areas

The first area was aimed at automating manual diagnostics done by the Wireshark tool. For this purpose, decision trees were used in which the knowledge database is stored. According to the specified criteria, specific events are searched for in the source data when traversing the tree. Based on whether these events are found, the process continues with the next tree node. This process is repeated until a tree's leaf node containing the description of the problem is found.

The second area was focused on the fact that some services may be unavailable to users due to an ongoing network attack. Unlike the existing IDS tool, the created method does not work on a packet or flow basis but can search for events in groups of packets defined by any value. This flexible grouping makes it possible to detect even very complex attacks.

The next area's goal was to create a mechanism that would look for specific events in the graphical data just as the administrator would look for them manually. However, instead of using image algorithms, chart data is converted into strings. The method is using regular expressions to search inside these strings. In contrast to existing solutions, a method based not only on the detection of outlier values is created but seeks out specific patterns having a specific meaning.

Another area deals with the possibility of learning previous diagnostic results and repeating them for the problems with the same behavior in the future. To learn the previous results, automata were used that describe the behavior of individual application protocols. If the administrator detects and diagnoses a new error, it is possible to extend the protocol

model by this error. If the same error occurs again, the administrator can quickly use the learned model instead of a complex diagnosis.

The last but one area focused on the data that is entered by the end-users of the computer network. If a user configures a network service, he types some configuration data on the keyboard. During the typing process, a typo error can occur inside the data. By using natural language processing techniques, it is possible to detect such an error and propose an adequate correction.

The last research area was different from the previous areas. In this case, the research was not focused on a new automatic diagnostic method, but on a visualization of network data. The goal was to allow quick top-level analysis of network data that can help administrators to understand what is happening in the network.

The list of research areas with belonging papers to these areas is shown in Table 4.1. All proposed methods were implemented in the form of proof-of-concept software and tested on a sample of test data. Together with the results, these methods were published in the form of reviewed research papers.

Research area	**Method description**	**Paper number and title**
Simulating real administrator diagnostic steps	Using decision trees that simulate manual diagnostic steps of the administrator by searching for specific events in a predefined order	I. Network Diagnostics Using Passive Network Monitoring and Packet Analysis
		II. Using Rule-Based Decision Trees for Automatic Passive Diagnostics of the Network Problems
Looking for security events	Searching for specific values and counting occurrences in customizable groups of packets	III. Automating Network Security Analysis at Packet-level by using Rule-based Engine
Simulating chart analysis of a real administrator	Converting data into string representation in which the visual patterns are searched for	IV. Pattern Detection Based Network Diagnostics
Diagnostic based on previous analysis	Learning automata from the previous analysis that describes protocols behavior and are used for future diagnostics	V. Using Network Traces to Generate Models for Automatic Network Application Protocols Diagnostics
		VI. Application Error Detection in Networks by Protocol Behavior Model
Looking for errors inside end-user data	Using algorithms from natural language processing to find typo errors and propose the correction candidate	VII. Network Problem Diagnostics using Typographic Error Correction
Network Traffic Top-Level Visual Analysis	Using a funnel filtering approach for flexible top-level data analysis.	VIII. PCAPFunnel: A Tool for Rapid Exploration of Packet Capture Files

Table 4.1: Summary of the research areas and papers that belong to those areas.

4.2 Summary of Research Objectives

This section summarizes the contribution by discussing on meeting research objectives that were introduced in Chapter 1 given by ultimate research goal, which was to create an easy-to-use extensible computer network error diagnosing system not limited to a small protocol set or a single error type, that requires only a small amount of captured network data without affecting the monitored networks. Based on the results, it may be concluded that the initial research goal was fulfilled.

Objective1: *Results of all created methods must be correct and accurate. Only in that situation, the methods can be trusted by network administrators and deployed in their networks.*

All presented diagnostic methods have been published in attached research papers. These papers contain an evaluation of those methods which vary from testing of simple errors inside the testing environment up to testing complex errors in a successful international commercial product.

Objective2: *The user of the system, a network administrator, needs to be able to easily extend the set of detectable errors. Because the usual administrator is not a programmer, the extension can not be based on source code updates.*

Easy extensibility is provided by using a set of configuration files by each diagnostic method. The methods are reading and executing instructions written inside those files. The configurations have a declarative format based on the Wireshark display language and can be added or updated without any change of tool implementation. Network applications commonly use the declarative format, and therefore the administrators are used to it. The Wireshark display language makes the configuration even more comfortable as they use the same language within the Wireshark and can specify what they want to do based on the previous knowledge with Wireshark.

Objective3: *Each well known and commonly used protocol needs to be supported. There should also be no difference between binary or text protocols.*

By using the TShark tool for processing and converting the network data, hundreds of network protocols are supported. As the TShark uses the same dissectors as the Wireshark, support of new protocols is added very quickly by the developers or the community.

Objective4: *The created methods can not expect a huge amount of network traffic, which they can analyze. Methods should expect the files within the tens of megabytes.*

None of the proposed methods uses a huge amount of data to create a baseline to diagnose possible errors. As described in each included paper, the methods expect only a small amount of network traffic containing only a subset of network traffic, e.g., traffic to the SMTP server.

Objective5: *During the diagnostic process, no traffic can be sent inside the monitored network. The methods need to work only with the provided data.*

All the proposed methods are using the passive approach to diagnose the problem. The main data source is the captured network traffic. However, as was stated inside the papers, the same approach can be used to diagnose from other passive sources such as log files or NetFlow records.

4.3 Software Outcome

One of the main achievements of this work is that the primary method of diagnostics based on decision trees has already been successfully integrated into a globally available commercial product called Flowmon Packet Investigator[1] (FPI), developed and managed by Flowmon Networks. Figure 4.1 shows the output from the product. Work is currently underway to integrate a method for data security analysis, and integration of other methods is expected in the future.

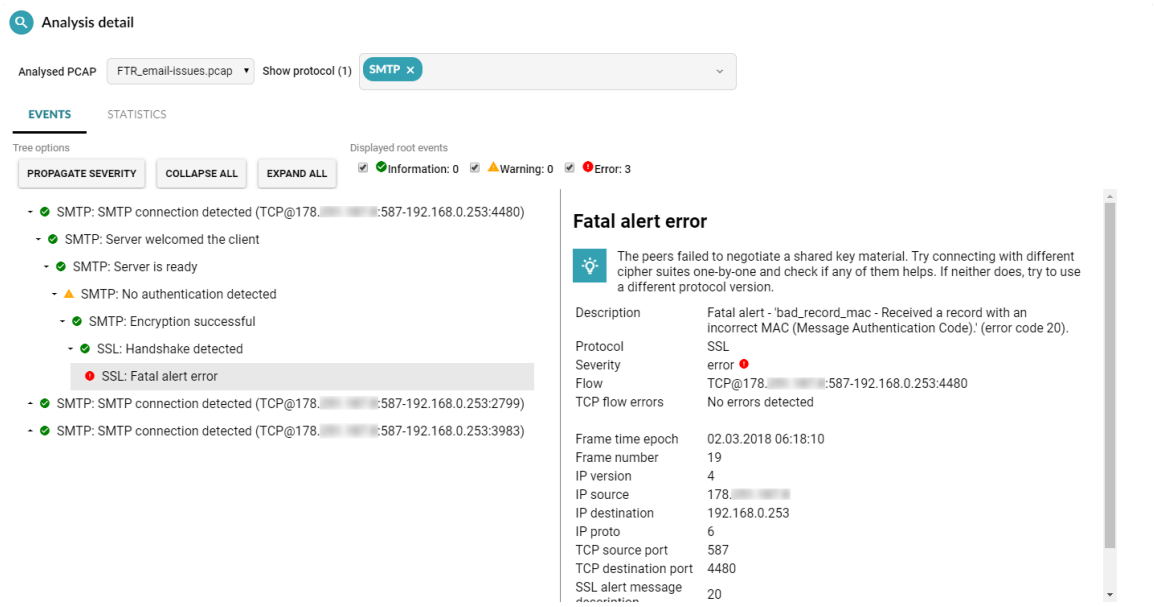

Figure 4.1: The output from the Flowmon Packet Investigator that describes detected error.

The FPI allows network administrators to upload a PCAP file, manually start capturing data on the probes placed inside the network, or automatically capture data inside the network based on the detection from the anomaly detection system. All these three options together create a powerful utility able to diagnose network problems based on different circumstances. The FPI has a modular structure that allows the execution of several diagnostic methods simultaneously and displays the results inside the one webpage window.

Currently, when writing this thesis, there is an ongoing project called *Streamlining cybersecurity incident analyses (SECURIAN)* supported by Technology Agency of the Czech

[1] https://www.flowmon.com/en/products/software-modules/packet-investigator

Republic that, in addition to other project goals, aims to integrate the results from the research area "Network Traffic Top-Level Visual Analysis" to the Flowmon Collector[2].

4.4 Towards a Practical Solution

The currently planned further work consists of the continuous integration of each developed method into the already mentioned Flowmon Packet Investigator tool. Also, several improvements and possible sequels have been identified:

- Accelerate processing of the data by the TShark tool. TShark itself cannot be run in parallel because packet processing may depend on any previous packet. By creating proper heuristics, it would be possible to split the input data into several independent smaller ones, each of which would be processed by an independent TShark process.
- All created methods are based on a deterministic approach, where events are detected based on exact rules. In some cases, however, it is not easy to determine for 100% that something is true or not. Therefore it makes sense to create methods using probabilistic techniques, such as fuzzy logic.
- The quality of the input data affects the quality of the diagnostic results. In the actual implementation, if some packets are missing inside the data, the general warning message that the results may not be accurate is being generated. However, the actual detection only works with missing TCP segments, and it is not inspected which data are missing and whether it affects the results. This detection can be improved by involving some machine learning techniques to ignore packet losses, which are not affecting the diagnostic results and also to be able to detect missing data in non-TCP data as well.
- In the current architecture, it is assumed that the administrator will ensure that the data are combined into one data set in the case of multiple sources. However, it might make sense to analyze the data distributively and only centralize the results from the individual locations.
- The implemented methods do not allow adaptation to a specific user (administrator). By analyzing the behavior of the users of the created tools, it would be possible to give more priority to some results or to hide some worthless results.

In addition to the mentioned items, it is always possible to continue with research by developing new diagnostic methods as new technologies and protocols evolve.

[2] https://www.flowmon.com/en/products/appliances/netflow-collector

Bibliography

[1] Bhavish Agarwal, Ranjita Bhagwan, Tathagata Das, Siddharth Eswaran, Venkata N Padmanabhan, and Geoffrey M Voelker. "NetPrints: Diagnosing Home Network Misconfigurations Using Shared Knowledge." In: *NSDI*. Vol. 9. 2009, pp. 349–364.

[2] Bhavish Aggarwal, Ranjita Bhagwan, Lorenzo De Carli, Venkat Padmanabhan, and Krishna Puttaswamy. "Deja vu: fingerprinting network problems". In: *Proceedings of the Seventh Conference on Emerging Networking Experiments and Technologies*. 2011, pp. 1–12.

[3] Dmytro Alekseev and Vladimir Sayenko. "Proactive fault detection in computer networks". In: *2014 First International Scientific-Practical Conference Problems of Infocommunications Science and Technology*. IEEE. 2014, pp. 90–91.

[4] Essam S Ali and MG Darwish. "Diagnosing network faults using bayesian and case-based reasoning techniques". In: *2007 International Conference on Computer Engineering & Systems*. IEEE. 2007, pp. 145–150.

[5] John D Andrews and Sarah J Dunnett. "Event-tree analysis using binary decision diagrams". In: *IEEE Transactions on Reliability* 49.2 (2000), pp. 230–238.

[6] João Antunes and Nuno Neves. "Automatically complementing protocol specifications from network traces". In: *Proceedings of the 13th European workshop on dependable computing*. 2011, pp. 87–92.

[7] Tanapat Anusas-Amornkul. "A network root cause analysis and repair system". In: *2018 6th International symposium on computational and business intelligence (ISCBI)*. IEEE. 2018, pp. 69–73.

[8] Pallavi Asrodia and Hemlata Patel. "Analysis of various packet sniffing tools for network monitoring and analysis". In: *International Journal of Electrical, Electronics and Computer Engineering* 1.1 (2012), pp. 55–58.

[9] Tariq Assaf and Joanne Bechta Dugan. "Build better diagnostic decision trees". In: *IEEE instrumentation & measurement magazine* 8.3 (2005), pp. 48–53.

[10] Paramvir Bahl, Ranveer Chandra, Albert Greenberg, Srikanth Kandula, David A Maltz, and Ming Zhang. "Towards highly reliable enterprise network services via inference of multi-level dependencies". In: *ACM SIGCOMM Computer Communication Review* 37.4 (2007), pp. 13–24.

[11] Julian Bangert and Nickolai Zeldovich. "Nail: A practical interface generator for data formats". In: *2014 IEEE Security and Privacy Workshops*. IEEE. 2014, pp. 158–166.

[12] Paul Barford, Nick Duffield, Amos Ron, and Joel Sommers. "Network performance anomaly detection and localization". In: *IEEE INFOCOM 2009*. IEEE. 2009, pp. 1377–1385.

[13] Timothy J Berners-Lee. *Information management: A proposal*. Tech. rep. 1989.

[14] Arpád Beszédes. "Investigating Fault Localization Techniques from Other Disciplines for Software Engineering". In: (2019).

[15] Alina Beygelzimer, Mark Brodie, Sheng Ma, and Irina Rish. "Test-based diagnosis: Tree and matrix representations". In: *2005 9th IFIP/IEEE International Symposium on Integrated Network Management, 2005. IM 2005.* IEEE. 2005, pp. 529–542.

[16] Monowar H Bhuyan, Dhruba Kumar Bhattacharyya, and Jugal K Kalita. "Network anomaly detection: methods, systems and tools". In: *Ieee communications surveys & tutorials* 16.1 (2013), pp. 303–336.

[17] Álvaro Brandón, Marc Solé, Alberto Huélamo, David Solans, Maria S Pérez, and Victor Muntés-Mulero. "Graph-based root cause analysis for service-oriented and microservice architectures". In: *Journal of Systems and Software* 159 (2020), p. 110432.

[18] Daniela Brauckhoff, Xenofontas Dimitropoulos, Arno Wagner, and Kavè Salamatian. "Anomaly extraction in backbone networks using association rules". In: *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*. 2009, pp. 28–34.

[19] Kenneth L Calvert, W Keith Edwards, Nick Feamster, Rebecca E Grinter, Ye Deng, and Xuzi Zhou. "Instrumenting home networks". In: *ACM SIGCOMM Computer Communication Review* 41.1 (2011), pp. 84–89.

[20] Dinis Canastro, Ricardo Rocha, Mário Antunes, Diogo Gomes, and Rui L Aguiar. "Root Cause Analysis in 5G/6G Networks". In: *2021 8th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE. 2021, pp. 217–224.

[21] Mike Chen, Alice X Zheng, Jim Lloyd, Michael I Jordan, and Eric Brewer. "Failure diagnosis using decision trees". In: *International Conference on Autonomic Computing, 2004. Proceedings.* IEEE. 2004, pp. 36–43.

[22] Ming Chen, Runqing Zhou, Rui Zhang, and Xianzhong Zhu. "Application of artificial neural network to failure diagnosis on process industry equipments". In: *2010 Sixth International Conference on Natural Computation*. Vol. 3. IEEE. 2010, pp. 1190–1193.

[23] Maggie X Cheng and Wei Biao Wu. "Data analytics for fault localization in complex networks". In: *IEEE Internet of Things Journal* 3.5 (2015), pp. 701–708.

[24] Edward Chuah, Arshad Jhumka, Sai Narasimhamurthy, John Hammond, James C Browne, and Bill Barth. "Linking resource usage anomalies with system failures from cluster log data". In: *2013 IEEE 32nd International Symposium on Reliable Distributed Systems*. IEEE. 2013, pp. 111–120.

[25] Paolo Milani Comparetti, Gilbert Wondracek, Christopher Kruegel, and Engin Kirda. "Prospex: Protocol specification extraction". In: *2009 30th IEEE Symposium on Security and Privacy*. IEEE. 2009, pp. 110–125.

[26] Arantxa Contreras-Valdes, Juan P Amezquita-Sanchez, David Granados-Lieberman, and Martin Valtierra-Rodriguez. "Predictive data mining techniques for fault diagnosis of electric equipment: A review". In: *Applied Sciences* 10.3 (2020), p. 950.

[27] Weidong Cui, Jayanthkumar Kannan, and Helen J Wang. "Discoverer: Automatic Protocol Reverse Engineering from Network Traces." In: *USENIX Security Symposium*. 2007, pp. 1–14.

[28] Amogh Dhamdhere, Renata Teixeira, Constantine Dovrolis, and Christophe Diot. "NetDiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data". In: *Proceedings of the 2007 ACM CoNEXT conference*. 2007, pp. 1–12.

[29] Akalanka Mailewa Dissanayaka, Susan Mengel, Lisa Gittner, and Hafiz Khan. "Vulnerability prioritization, root cause analysis, and mitigation of secure data analytic framework implemented with mongodb on singularity linux containers". In: *Proceedings of the 2020 the 4th International Conference on Compute and Data Analysis*. 2020, pp. 58–66.

[30] Ibrahim Ali Ibrahim Diyeb, Anwar Saif, and Nagi Ali Al-Shaibany. "Ethical network surveillance using packet sniffing tools: A comparative study". In: *International Journal of Computer Network and Information Security* 11.7 (2018), p. 12.

[31] Mentari Djatmiko, Dominik Schatzmann, Arik Friedman, Xenofontas Dimitropoulos, and Roksana Boreli. "Privacy preserving distributed network outage monitoring". In: *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2013, pp. 69–70.

[32] Changyu Dong and Naranker Dulay. "Argumentation-based fault diagnosis for home networks". In: *Proceedings of the 2nd ACM SIGCOMM Workshop on Home Networks*. 2011, pp. 37–42.

[33] Ayush Dusia and Adarshpal S Sethi. "Recent advances in fault localization in computer networks". In: *IEEE Communications Surveys & Tutorials* 18.4 (2016), pp. 3030–3051.

[34] Amanuel Ayde Ergado. "Self learning computer troubleshooting expert system". In: *International Journal of Artificial Intelligence & Applications (IJAIA)* 7.1 (2016), pp. 45–58.

[35] Min Feng and Rajiv Gupta. "Learning universal probabilistic models for fault localization". In: *Proceedings of the 9th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*. 2010, pp. 81–88.

[36] Zhe Feng and Alban Grastien. "Model Based Diagnosis of Timed Automata with Model Checkers". In: ().

[37] Xiaoyu Fu, Rui Ren, Sally A McKee, Jianfeng Zhan, and Ninghui Sun. "Digging deeper into cluster system logs for failure prediction and root cause diagnosis". In: *2014 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2014, pp. 103–112.

[38] Maurice Gagnaire, Felipe Diaz, Camille Coti, Christophe Cerin, Kazuhiko Shiozaki, Yingjie Xu, Pierre Delort, Jean-Paul Smets, Jonathan Le Lous, Stephen Lubiarz, et al. "Downtime statistics of current cloud solutions". In: *International Working Group on Cloud Computing Resiliency, Tech. Rep* (2012).

[39] Mohammad Sadeq Garshasbi. "Fault localization based on combines active and passive measurements in computer networks by ant colony optimization". In: *Reliability Engineering & System Safety* 152 (2016), pp. 205–212.

[40] Glen Gibb, George Varghese, Mark Horowitz, and Nick McKeown. "Design principles for packet parsers". In: *Architectures for Networking and Communications Systems*. IEEE. 2013, pp. 13–24.

[41] Christos Gkantsidis and Hitesh Ballani. *Network management as a service*. Tech. rep. Microsoft Research, Technical Report MSR-TR-2010-83, 2010.

[42] Eric Golden and John W Coffey. "A tool to automate generation of wireshark dissectors for a proprietary communication protocol". In: *The 6th International Conference on Complexity, Informatics and Cybernetics, IMCIC*. 2015.

[43] Salah Gontara, Amine Boufaied, and Ouajdi Korbaa. "A unified approach for selecting probes and probing stations for fault detection and localization in computer networks". In: *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE. 2019, pp. 2071–2076.

[44] Tang Haina, Han Chunjing, and Ge Jingguo. "Applications of visualization technology for network security". In: *2017 IEEE Trustcom/BigDataSE/ICESS*. IEEE. 2017, pp. 1038–1042.

[45] Rick Hofstede, Pavel Čeleda, Brian Trammell, Idilio Drago, Ramin Sadre, Anna Sperotto, and Aiko Pras. "Flow monitoring explained: From packet capture to data analysis with netflow and ipfix". In: *IEEE Communications Surveys & Tutorials* 16.4 (2014), pp. 2037–2064.

[46] Demetris Hoplaros, Zahir Tari, and Ibrahim Khalil. "Data summarization for network traffic monitoring". In: *Journal of network and computer applications* 37 (2014), pp. 194–205.

[47] Wei Huang. "A practical guide of troubleshooting IEC 61850 GOOSE communication". In: *2017 70th Annual Conference for Protective Relay Engineers (CPRE)*. IEEE. 2017, pp. 1–8.

[48] Shahram Jamali and Mohammad Sadeq Garshasbi. "Fault localization algorithm in computer networks by employing a genetic algorithm". In: *Journal of ExpErimEntal & thEorEtical artificial intElligEncE* 29.1 (2017), pp. 157–174.

[49] Umar Javed, Italo Cunha, David Choffnes, Ethan Katz-Bassett, Thomas Anderson, and Arvind Krishnamurthy. "PoiRoot: Investigating the root cause of interdomain path changes". In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 183–194.

[50] Rui Jia, Sherif Abdelwahed, and Abdelkarim Erradi. "Towards proactive fault management of enterprise systems". In: *2015 International Conference on Cloud and Autonomic Computing*. IEEE. 2015, pp. 21–32.

[51] Cheng Jiang, Weilin Deng, and Daowen Qiu. "Fault Diagnosis in Unknown Discrete Event Systems via Critical Tree". In: *2019 Chinese Control And Decision Conference (CCDC)*. IEEE. 2019, pp. 1846–1851.

[52] Yu Jin, Nick Duffield, Alexandre Gerber, Patrick Haffner, Subhabrata Sen, and Zhi-Li Zhang. "Nevermind, the problem is already fixed: proactively detecting and troubleshooting customer dsl problems". In: *Proceedings of the 6th International COnference*. 2010, pp. 1–12.

[53] Diana Joumblatt, Jaideep Chandrashekar, Branislav Kveton, Nina Taft, and Renata Teixeira. "Predicting user dissatisfaction with internet application performance at end-hosts". In: *2013 Proceedings IEEE INFOCOM*. IEEE. 2013, pp. 235–239.

[54] Srikanth Kandula, Ratul Mahajan, Patrick Verkaik, Sharad Agarwal, Jitendra Padhye, and Paramvir Bahl. "Detailed diagnosis in enterprise networks". In: *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*. 2009, pp. 243–254.

[55] Soila P Kavulya, Scott Daniels, Kaustubh Joshi, Matti Hiltunen, Rajeev Gandhi, and Priya Narasimhan. "Draco: Statistical diagnosis of chronic problems in large distributed systems". In: *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*. IEEE. 2012, pp. 1–12.

[56] Marie Kiermeier, Sebastian Feld, and Claudia Linnhoff-Popien. "Root cause analysis for global anomalous events in self-organizing industrial systems". In: *2017 IEEE 21st International Conference on Intelligent Engineering Systems (INES)*. IEEE. 2017, pp. 000163–000168.

[57] Kyung-Hwa Kim, Hyunwoo Nam, Jin-Hyung Park, and Henning Schulzrinne. "MoT: a collaborative network troubleshooting platform for the internet of things". In: *2014 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE. 2014, pp. 3438–3443.

[58] Kyung-Hwa Kim, Hyunwoo Nam, Vishal Singh, Daniel Song, and Henning Schulzrinne. "DYSWIS: Crowdsourcing a home network diagnosis". In: *2014 23rd International conference on computer communication and networks (ICCCN)*. IEEE. 2014, pp. 1–10.

[59] Myunghwan Kim, Roshan Sumbaly, and Sam Shah. "Root cause detection in a service-oriented architecture". In: *ACM SIGMETRICS Performance Evaluation Review* 41.1 (2013), pp. 93–104.

[60] Siheung Kim, Seong jin Ahn, Jinwok Chung, Ilsung Hwang, Sunghe Kim, Minki No, and Seungchung Sin. "A rule based approach to network fault and security diagnosis with agent collaboration". In: *Artificial Intelligence and Simulation: 13th International Conference on AI, Simulation, Planning in High Autonomy Systems, AIS 2004, Jeju Island, Korea, October 4-6, 2004, Revised Selected Papers 13*. Springer. 2005, pp. 597–606.

[61] Tatsuaki Kimura, Akio Watanabe, Tsuyoshi Toyono, and Keisuke Ishibashi. "Proactive failure detection learning generation patterns of large-scale network logs". In: *IEICE Transactions on Communications* 102.2 (2019), pp. 306–316.

[62] Ramana Rao Kompella, Jennifer Yates, Albert Greenberg, and Alex C Snoeren. "Fault localization via risk modeling". In: *IEEE Transactions on Dependable and Secure Computing* 7.4 (2009), pp. 396–409.

[63] Inès Ben Kraiem, Faiza Ghozzi, André Péninou, Geoffrey Roman-Jimenez, and Olivier Teste. "Human-interpretable rules for anomaly detection in time-series". In: *INTERNATIONAL CONFERENCE ON EXTENDING DATABASE TECHNOLOGY*. OpenProceedings. org. 2021, pp. 457–462.

[64] Tammo Krueger, Hugo Gascon, Nicole Krämer, and Konrad Rieck. "Learning stateful models for network honeypots". In: *Proceedings of the 5th ACM workshop on Security and artificial intelligence*. 2012, pp. 37–48.

[65] Dmitriy Kuptsov, Boris Nechaev, Prabhu Patil, and Andrey Khurri. "ORCE: Online Root Cause Estimator for TCP". In: (2009).

[66] Martin Laštovička, Martin Husák, and Lukáš Sadlek. “Network monitoring and enumerating vulnerabilities in large heterogeneous networks”. In: *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE. 2020, pp. 1–6.

[67] Sihyung Lee and Hyong S Kim. “End-user perspectives of Internet connectivity problems”. In: *Computer Networks* 56.6 (2012), pp. 1710–1722.

[68] Sihyung Lee, Kyriaki Levanti, and Hyong S Kim. “Network monitoring: Present and future”. In: *Computer Networks* 65 (2014), pp. 84–98.

[69] Ma łgorzata Steinder and Adarshpal S Sethi. “A survey of fault localization techniques in computer networks”. In: *Science of computer programming* 53.2 (2004), pp. 165–194.

[70] Bingdong Li, Jeff Springer, George Bebis, and Mehmet Hadi Gunes. “A survey of network flow applications”. In: *Journal of Network and Computer Applications* 36.2 (2013), pp. 567–581.

[71] Yuxing Li, Hu Zheng, Chengqiang Huang, Ke Pei, Jinghui Li, and Longbo Huang. “Terminator: An Efficient and Light-weight Fault Localization Framework”. In: *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2020, pp. 580–585.

[72] Lei Liu, Xiaolong Jin, Geyong Min, and Li Xu. “Real-time diagnosis of network anomaly based on statistical traffic analysis”. In: *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE. 2012, pp. 264–270.

[73] Lu Lu, Zhengguo Xu, Wenhai Wang, and Youxian Sun. “A new fault detection method for computer networks”. In: *Reliability Engineering & System Safety* 114 (2013), pp. 45–51.

[74] J Lunze and P Supavatanakul. “Application of timed automata to the diagnosis of the DAMADICS benchmark problem”. In: *IFAC Proceedings Volumes* 36.5 (2003), pp. 1083–1088.

[75] Chuanfei Luo, Jun Sun, and Hongkai Xiong. “Monitoring and troubleshooting in operational IP-TV system”. In: *IEEE Transactions on Broadcasting* 53.3 (2007), pp. 711–718.

[76] Ming Luo, Danhong Zhang, GeokHong Phua, Lihui Chen, and Danwei Wang. “An interactive rule based event management system for effective equipment troubleshooting”. In: *IECON 2011-37th Annual Conference of the IEEE Industrial Electronics Society*. IEEE. 2011, pp. 2329–2334.

[77] Mohammed Madi, Fidaa Jarghon, Yousef Fazea, Omar Almomani, and Adeeb Saaidah. “Comparative analysis of classification techniques for network fault management”. In: *Turkish Journal of Electrical Engineering and Computer Sciences* 28.3 (2020), pp. 1442–1457.

[78] Ratul Mahajan, Neil Spring, David Wetherall, and Thomas Anderson. “User-level Internet path diagnosis”. In: *ACM SIGOPS Operating Systems Review* 37.5 (2003), pp. 106–119.

[79] Ajay Mahimkar, Jennifer Yates, Yin Zhang, Aman Shaikh, Jia Wang, Zihui Ge, and Cheng Tien Ee. “Troubleshooting chronic conditions in large IP networks”. In: *Proceedings of the 2008 ACM CoNEXT Conference*. 2008, pp. 1–12.

[80] Ajay Anil Mahimkar, Zihui Ge, Aman Shaikh, Jia Wang, Jennifer Yates, Yin Zhang, and Qi Zhao. "Towards automated performance diagnosis in a large IPTV network". In: *ACM SIGCOMM Computer Communication Review* 39.4 (2009), pp. 231–242.

[81] Leonardo Mariani, Cristina Monni, Mauro Pezzé, Oliviero Riganelli, and Rui Xin. "Localizing faults in cloud systems". In: *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE. 2018, pp. 262–273.

[82] Angelos K Marnerides, Alberto Schaeffer-Filho, and Andreas Mauthe. "Traffic anomaly diagnosis in Internet backbone networks: A survey". In: *Computer Networks* 73 (2014), pp. 224–243.

[83] Eloy Martinez, Enda Fallon, Sheila Fallon, and MingXue Wang. "Cadmant: Context anomaly detection for maintenance and network troubleshooting". In: *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE. 2015, pp. 1017–1022.

[84] Maha Mdini, Gwendal Simon, Alberto Blanc, and Julien Lecoeuvre. "ARCD: a solution for root cause diagnosis in mobile networks". In: *2018 14th International Conference on Network and Service Management (CNSM)*. IEEE. 2018, pp. 280–284.

[85] Venkat Mohan, YR Janardhan Reddy, and Kulshrestha Kalpana. "Active and passive network measurements: a survey". In: *International Journal of Computer Science and Information Technologies* 2.4 (2011), pp. 1372–1385.

[86] Karthik Nagaraj, Charles Edwin Killian, and Jennifer Neville. "Structured comparative analysis of systems logs to diagnose performance problems." In: *NSDI*. 1. 2012, p. 353.

[87] Mehdi Namdari, Hooshang Jazayeri-Rad, and Nader Nabhani. "Comparing the performance of two neural network methods in a process fault diagnosis system". In: *J. Basic Appl. Sci. Res.* 3.2 (2013), pp. 942–947.

[88] Stefano Nicastro. "Complexity and emergence in data networks". In: (2021).

[89] Byungchul Park, Young J Won, Hwanjo Yu, James Won-Ki Hong, Hong-Sun Noh, and Jang Jin Lee. "Fault detection in IP-based process control networks using data mining". In: *2009 IFIP/IEEE International Symposium on Integrated Network Management*. IEEE. 2009, pp. 211–217.

[90] Gaetano Pellegrino, Qin Lin, Christian Hammerschmidt, and Sicco Verwer. "Learning behavioral fingerprints from netflows using timed automata". In: *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE. 2017, pp. 308–316.

[91] Sylvain Piechowiak and Joaquin Rodriguez. "The localization and correction of errors in models: A constraint-based approach". In: *Applied Intelligence* 23 (2005), pp. 153–164.

[92] Rahul Potharaju and Navendu Jain. "Demystifying the dark side of the middle: A field study of middlebox failures in datacenters". In: *Proceedings of the 2013 conference on Internet measurement conference*. 2013, pp. 9–22.

[93] M Procházka, D Macko, and K Jelemenská. "IP networks diagnostic communication generator". In: *2017 15th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. IEEE. 2017, pp. 1–6.

[94] Henry C Pusey and Paul L Howard. “An historical view of mechanical failure prevention technology”. In: *Sound and Vibration* (2008), p. 11.

[95] Tongqing Qiu, Zihui Ge, Dan Pei, Jia Wang, and Jun Xu. “What happened in my network: mining network events from router syslogs”. In: *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. 2010, pp. 472–484.

[96] Xiang Rao, Huaimin Wang, Dianxi Shi, Zhenbang Chen, Hua Cai, Qi Zhou, and Tingtao Sun. “Identifying faults in large-scale distributed systems by filtering noisy error logs”. In: *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE. 2011, pp. 140–145.

[97] Gianluca Reali, Mauro Femminella, and Luca Monacelli. “Probabilistic codebook-based fault localization in data networks”. In: *IEEE Transactions on Network and Service Management* 15.2 (2018), pp. 567–581.

[98] Eleni Rozaki. “Design and implementation for automated network troubleshooting using data mining”. In: *arXiv preprint arXiv:1506.00108* (2015).

[99] A Samhat, R Skehill, and Z Altman. “Automated troubleshooting in WLAN networks”. In: *2007 16th IST Mobile and Wireless Communications Summit*. IEEE. 2007, pp. 1–4.

[100] Ricardo L dos Santos, Juliano A Wickboldt, Roben C Lunardi, Bruno L Dalmazo, Lisandro Z Granville, Luciano P Gaspary, Claudio Bartolini, and Marianne Hickey. “A solution for identifying the root cause of problems in it change management”. In: *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*. IEEE. 2011, pp. 586–593.

[101] Eunsoo Seo, Gulustan Dogan, Tarek Abdelzaher, and Theodore Brown. “Root cause diagnosis in error-propagating networks”. In: *Security and Communication Networks* 9.11 (2016), pp. 1297–1308.

[102] Matti Siekkinen, Guillaume Urvoy-Keller, Ernst W Biersack, and Denis Collange. “A root cause analysis toolkit for TCP”. In: *Computer Networks* 52.9 (2008), pp. 1846–1858.

[103] Leslie F Sikos. “Packet analysis for network forensics: A comprehensive survey”. In: *Forensic Science International: Digital Investigation* 32 (2020), p. 200892.

[104] Leslie F Sikos. “Packet analysis for network forensics: A comprehensive survey”. In: *Forensic Science International: Digital Investigation* 32 (2020), p. 200892.

[105] Su Myat Marlar Soe and May Paing Paing Zaw. “Design and implementation of rule-based expert system for fault management”. In: *International Journal of Computer and Information Engineering* 2.12 (2008), pp. 4022–4027.

[106] Marc Solé, Victor Muntés-Mulero, Annie Ibrahim Rana, and Giovani Estrada. “Survey on models and techniques for root-cause analysis”. In: *arXiv preprint arXiv:1701.08546* (2017).

[107] Anna Cinzia Squicciarini, Giuseppe Petracca, William G Horne, and Aurnob Nath. “Situational awareness through reasoning on network incidents”. In: *Proceedings of the 4th ACM conference on Data and application security and privacy*. 2014, pp. 111–122.

[108] Srinikethan Madapuzi Srinivasan, Tram Truong-Huu, and Mohan Gurusamy. "Machine learning-based link fault identification and localization in complex networks". In: *IEEE Internet of Things Journal* 6.4 (2019), pp. 6556–6566.

[109] Srikanth Sundaresan, Yan Grunenberger, Nick Feamster, Dina Papagiannaki, Dave Levin, and Renata Teixeira. *WTF? Locating performance problems in home networks*. Tech. rep. Georgia Institute of Technology, 2013.

[110] P Supavatanakul, J Lunze, V Puig, and J Quevedo. "Diagnosis of timed automata: Theory and application to the DAMADICS actuator benchmark problem". In: *Control Engineering Practice* 14.6 (2006), pp. 609–619.

[111] Jakub Svoboda, Ibrahim Ghafir, Vaclav Prenosil, et al. "Network monitoring approaches: An overview". In: *Int J Adv Comput Netw Secur* 5.2 (2015), pp. 88–93.

[112] Masatoshi Takano. "ICS Cybersecurity incident response and the troubleshooting process". In: *2014 Proceedings of the SICE Annual Conference (SICE)*. IEEE. 2014, pp. 827–832.

[113] Chung-hao Tan. "Failure Diagnosis for Configuration Problem in Storage System". In: *IBM Almaden Reseach Center* ().

[114] Anuja Tayal, Neha Sharma, Neminath Hubballi, and Maitreya Natu. "Traffic dynamics-aware probe selection for fault detection in networks". In: *Journal of Network and Systems Management* 28 (2020), pp. 1055–1084.

[115] Stefano Traverso, Edion Tego, Eike Kowallik, Stefano Raffaglio, Andrea Fregosi, Marco Mellia, and Francesco Matera. "Exploiting hybrid measurements for network troubleshooting". In: *2014 16th International Telecommunications Network Strategy and Planning Symposium (Networks)*. IEEE. 2014, pp. 1–6.

[116] Alex Ulmer, David Sessler, and Jörn Kohlhammer. "Netcapvis: Web-based progressive visual analytics for network packet captures". In: *2019 IEEE Symposium on Visualization for Cyber Security (VizSec)*. IEEE. 2019, pp. 1–10.

[117] Tong Van, Hai Anh Tran, Sami Souihi, and Abdelhamid MELLOUK. "Network troubleshooting: survey, taxonomy and challenges". In: *2018 International Conference on Smart Communications in Network Technologies (SaCoNeT)*. IEEE. 2018, pp. 165–170.

[118] Angela M Vargas-Arcila, Juan Carlos Corrales, Araceli Sanchis, and Álvaro Rendón Gallón. "Peripheral diagnosis for propagated network faults". In: *Journal of Network and Systems Management* 29.2 (2021), p. 14.

[119] Chengwel Wang, Soila P Kavulya, Jiaqi Tan, Liting Hu, Mahendra Kutare, Mike Kasick, Karsten Schwan, Priya Narasimhan, and Rajeev Gandhi. "Performance troubleshooting in data centers: an annotated bibliography?" In: *ACM SIGOPS Operating Systems Review* 47.3 (2013), pp. 50–62.

[120] Hanzhang Wang, Zhengkai Wu, Huai Jiang, Yichao Huang, Jiamu Wang, Selcuk Kopru, and Tao Xie. "Groot: An event-graph-based approach for root cause analysis in industrial settings". In: *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2021, pp. 419–429.

[121] RiXin Wang, Yang Jin, and MinQiang Xu. "Approach of fault diagnosis based on similarity degree matching distance function". In: *Science China Technological Sciences* 56 (2013), pp. 2709–2720.

[122] Ruyan Wang, Dapeng Wu, Yang Li, Xiong Yu, Zhao Hui, and Keping Long. "Knight's tour-based fast fault localization mechanism in mesh optical communication networks". In: *Photonic Network Communications* 23 (2012), pp. 123–129.

[123] Yien Wang and Jianhua Yang. "Ethical hacking and network defense: choose your best network vulnerability scanning tool". In: *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. IEEE. 2017, pp. 110–113.

[124] Marco Weiss, Simon Bauer, and Benedikt Jaeger. "Optimization of Decision Trees for TCP Performance Root Cause Analysis". In: *Network* 67 (2019).

[125] Chathuranga Widanapathirana, Jonathan Li, Y Ahmet Sekercioglu, Milosh Ivanovich, and Paul Fitzpatrick. "Intelligent automated diagnosis of client device bottlenecks in private clouds". In: *2011 Fourth IEEE International Conference on Utility and Cloud Computing*. IEEE. 2011, pp. 261–266.

[126] Gilbert Wondracek, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, and Scuola Superiore S Anna. "Automatic Network Protocol Analysis." In: *NDSS*. Vol. 8. Citeseer. 2008, pp. 1–14.

[127] Xin Wu, Daniel Turner, Chao-Chih Chen, David A Maltz, Xiaowei Yang, Lihua Yuan, and Ming Zhang. "NetPilot: Automating datacenter network failure mitigation". In: *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. 2012, pp. 419–430.

[128] Andreas Wundsam, Amir Mehmood, Anja Feldmann, and Olaf Maennel. "Network troubleshooting with mirror VNets". In: *2010 IEEE Globecom Workshops*. IEEE. 2010, pp. 283–287.

[129] Ming-Ming Xiao, Shun-Zheng Yu, and Yu Wang. "Automatic network protocol automaton extraction". In: *2009 Third International Conference on Network and System Security*. IEEE. 2009, pp. 336–343.

[130] Chunmei Xu, Hao Zhang, Conghua Huang, and Daogang Peng. "Study of fault diagnosis based on probabilistic neural network for turbine generator unit". In: *2010 International Conference on Artificial Intelligence and Computational Intelligence*. Vol. 1. IEEE. 2010, pp. 275–279.

[131] Tao Xu, Dian Shen, Huanhuan Zhang, Runqun Xiong, and Jiahui Jin. "Fault diagnosis for the virtualized network in the cloud environment using reinforcement learning". In: *2019 IEEE International Conference on Smart Cloud (SmartCloud)*. IEEE. 2019, pp. 231–236.

[132] Tianyin Xu and Yuanyuan Zhou. "Systems approaches to tackling configuration errors: A survey". In: *ACM Computing Surveys (CSUR)* 47.4 (2015), pp. 1–41.

[133] He Yan, Lee Breslau, Zihui Ge, Dan Massey, Dan Pei, and Jennifer Yates. "G-rca: a generic root cause analysis platform for service quality management in large ip networks". In: *Proceedings of the 6th International COnference*. 2010, pp. 1–12.

[134] Yi Yang, Kieran McLaughlin, Tim Littler, Sakir Sezer, and HF Wang. "Rule-based intrusion detection system for SCADA networks". In: (2013).

[135] Xu Ye, Bo Yan, and Guanling Chen. “Providing diagnostic network feedback to end users on smartphones”. In: *2013 IEEE 32nd International Performance Computing and Communications Conference (IPCCC)*. IEEE. 2013, pp. 1–9.

[136] Mao Yi. “Computer Fault Diagnosis System Based on Artificial Intelligence”. In: *Revista Ibérica de Sistemas e Tecnologias de Informação* 16B (2015), p. 338.

[137] Chun Yuan, Ni Lao, Ji-Rong Wen, Jiwei Li, Zheng Zhang, Yi-Min Wang, and Wei-Ying Ma. “Automated known problem diagnosis with event traces”. In: *ACM SIGOPS Operating Systems Review* 40.4 (2006), pp. 375–388.

[138] Ding Yuan, Haohui Mai, Weiwei Xiong, Lin Tan, Yuanyuan Zhou, and Shankar Pasupathy. “Sherlog: error diagnosis by connecting clues from run-time logs”. In: *Proceedings of the fifteenth International Conference on Architectural support for programming languages and operating systems*. 2010, pp. 143–154.

[139] Hongyi Zeng, Peyman Kazemian, George Varghese, and Nick McKeown. “A survey on network troubleshooting”. In: *Technical Report Stanford/TR12-HPNG-061012, Stanford University, Tech. Rep.* (2012).

[140] Hongyi Zeng, Peyman Kazemian, George Varghese, and Nick McKeown. “Automatic test packet generation”. In: *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. 2012, pp. 241–252.

[141] Xin Zhang, Fanfu Zhou, Xinyu Zhu, Haiyang Sun, Adrian Perrig, Athanasios V Vasilakos, and Haibing Guan. “DFL: Secure and practical fault localization for datacenter networks”. In: *IEEE/ACM Transactions on Networking* 22.4 (2013), pp. 1218–1231.

[142] Ziming Zheng, Li Yu, Zhiling Lan, and Terry Jones. “3-dimensional root cause diagnosis via co-analysis”. In: *Proceedings of the 9th international conference on Autonomic computing*. 2012, pp. 181–190.

[143] Wang Zhenqi and Wang Xinyu. “Netflow based intrusion detection system”. In: *2008 International conference on multimedia and information technology*. IEEE. 2008, pp. 825–828.

[144] Donghao Zhou, Zheng Yan, Yulong Fu, and Zhen Yao. “A survey on network data collection”. In: *Journal of Network and Computer Applications* 116 (2018), pp. 9–23.

[145] Pengpeng Zhou, Yang Wang, Zhenyu Li, Gareth Tyson, Hongtao Guan, and Gaogang Xie. “Logchain: Cloud workflow reconstruction & troubleshooting with unstructured logs”. In: *Computer Networks* 175 (2020), p. 107279.

[146] Yanyan Zhuang, Eleni Gessiou, Steven Portzer, Fraida Fund, Monzur Muhammad, Ivan Beschastnikh, and Justin Cappos. “Netcheck: Network diagnoses from blackbox traces”. In: *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*. 2014, pp. 115–128.

Appendix A

Included Papers

A.1 Network Diagnostics Using Passive Network Monitoring and Packet Analysis

Authors: Holkovič Martin, Ing. (80%), Ryšavý Ondřej, doc. Ing., Ph.D. (20%)
Abstract: Finding a problem cause in network infrastructure is a complex task because a fault node may impair seemingly independent components. On the other hand, most communication protocols have built-in error detection mechanisms. In this paper, we propose to build a system that automatically diagnoses network services and applications by inspecting the network communication automatically. We model the diagnostic problem using a fault tree method and generate a set of rules that identify the symptoms and link them with possible causes. The administrators can extend these rules based on their experiences and the network configuration to automatize their routine tasks. We successfully deployed the proof-of-concept tool and found interesting future research topics.
Keywords: Network diagnostics, passive network monitoring, rule-based diagnostics, fault tree analysis, event-based diagnostics.
Published in: The Fifteenth International Conference on Networking and Services (ICNS-2019), Athens, Greece
Conference rating: N/A (Core), B4 (Qualis)
ISSN: 2308-4006
ISBN: 978-1-61208-711-5

Network Diagnostics Using Passive Network Monitoring and Packet Analysis

Martin Holkovič

CESNET a.l.e.
Zikova 1903/4
Prague 16000, CZ
Email: holkovic@cesnet.cz

Ondřej Ryšavý

Faculty of Information Technology
Brno University of Technology
Brno 61266, CZ
Email: rysavy@fit.vutbr.cz

***Abstract*—Finding a problem cause in network infrastructure is a complex task because a fault node may impair seemingly independent components. On the other hand, most communication protocols have built-in error detection mechanisms. In this paper, we propose to build a system that automatically diagnoses network services and applications by inspecting the network communication automatically. We model the diagnostic problem using a fault tree method and generate a set of rules that identify the symptoms and link them with possible causes. The administrators can extend these rules based on their experiences and the network configuration to automatize their routine tasks. We successfully deployed the proof-of-concept tool and found interesting future research topics.**

Keywords–Network diagnostics; passive network monitoring; rule-based diagnostics; fault tree analysis; event-based diagnostics.

I. Introduction

Network infrastructure and applications are complex, prone to cyber attacks, outages, performance problems, misconfiguration errors, and problems caused by software or hardware incompatibility. All these problems may affect network performance and user experience [1] which may cause fatal problems in critical networks (e.g., E-health, Vanet, Industrial IoT).

Many network administrators do not have the proper tools or knowledge to diagnose and fix network problems effectively, and they require an automated tool to diagnose these errors [2]. Zeng et al. [3] provide a short survey on network troubleshooting from the administrators' viewpoint identifying the most common network problems: *reachability problems, degraded throughput, high latency, and intermittent connectivity*. The consulted network administrators expressed the need for a network monitoring tool that would be able to identify such problems.

This paper proposes a system which creates diagnostic information only by performing passive network traffic monitoring and packet-level analysis. Previous research and development provided tools for helping administrators to diagnose faults [4] and performance problems [5]. However, these tools either require *installation of agents on hosts, active monitoring, or providing rich information about the environment*.

One of the most common ways of analyzing network traffic is by using a network packet analyzer (e.g., Wireshark). The analyzer works with captured network traffic (PCAP files) and displays structured information of layered protocols contained in every packet (encapsulated protocols, protocol fields). Administrators work with this information, check transferred content and compare the data with expected values. This process, done manually, is time-consuming and requires a good knowledge of network protocols and technologies.

The main contribution of this paper is a proposal of a tool for automatic diagnoses of network related problems from network communication only. Our approach tries to imitate a diagnostic process of a real administrator using the fault tree method and a popular packet parsing tool tshark. We have also implemented a proof-of-concept implementation to confirm the viability of the approach.

The paper is organized as follows. Section 2 defines the problem statement and research questions. Section 3 discusses related work and describes diagnostic approaches. Our solution consists of three stages and is introduced in Section 4. Section 5 instructs network administrators how to use our system (proof-of-concept) and shows how we model diagnostic knowledge. Finally, Section 6 is the conclusion which summarizes the current state and proposes future works.

II. Research Questions

Our primary goal is to design a system that infers possible causes accountable for network related problems, such as service unreachability or application errors. Offering a list of actions for fixing the errors' cause is the secondary and optional goal. All this information is gathered only from captured network communication.

In our work, we focus on enterprise networks that have complex networking topologies, usually consisting of heterogeneous devices. We expect that the administrators will collect network communication on appropriate places and validate its consistency before the analysis.

To achieve our goal, we need to find answers to the following research questions:

1) How to model different network faults in a suitable way for implementation in a diagnostic system? *Reachability, application specific, and device malfunctioning problems* can cause various networking issues. We need to have a unified approach for modeling these problems to identify the symptoms and link them with root causes.
2) What information should be extracted from the captured network communication to identify symptoms of failures?

In our case, we can passively access the communication in the monitored network and extract the necessary data to detect possible symptoms. An approach that can efficiently detect the symptoms in terms of precision and performance is needed.

3) How to identify the root cause of the problem, if we have a set of identified symptoms? The core part of the diagnostic engine is to apply knowledge gathered from observed symptoms to infer the possible root cause of the observed problem. The result should provide the information in sufficient detail. For instance, if the process on server crashed, then we would like to know this specific information instead of a more general explanation (e.g., a host failure has occurred).
4) What list of actions can we give to the administrator to fix the problems? Based on the observed symptoms and the root cause, the system should be able to provide fixing guidelines. These guidelines are supposed to be easy to understand even for an inexperienced administrator.

III. Related Work

A lot of research activities were dedicated to the diagnoses of network faults. Various methods were proposed for different network environments [4], in particular, home networks [6], enterprise networks [7]–[10], data centers [5], backbone and telecommunications networks [11], mobile networks [12], Internet of Things [13], Internet routing [14] and host reachability. Methods of network troubleshooting can be roughly divided into the following classes:

Active methods use traffic generators to send probe packets that can detect the availability of services or check the status of applications [15]. Usually, generators create diagnostic communication according to the test plan [7]. The responses are evaluated and provide diagnostic information that may help to reveal device misconfiguration or transient fail network states. Diagnostic probes introduce extra traffic, which may pose a problem for large installations [10]. Also, active methods may rely on the deployment of an agent within the environment to get information about the individual nodes [8].

Passive methods detect symptoms from existing data sources, e.g., traffic metadata [11], traffic capture files, network log files [14], performance counters. Passive methods can utilize the data provided by network monitoring systems.

Of course, the proposed systems also combine passive traffic monitoring to detect faults with active probing to determine the cause of failure. Identifying anomalies related to network faults and linking them with possible causes can be done by using one of the following approaches:

Inference-based approach uses a *model* to identify the dependence among components and to infer the faults using a collection of facts about the individual components [8], [16].

Rule-based approach uses *predefined rules* to diagnose faults [9]. The rules identify symptoms and determine how these contribute to the cause. The rules may be organized in a collaborative environment for sharing knowledge between administrators [6].

Classifier-based approach *requires training data* to learn the normal and faulty states. The classifier can identify a fault and its likely cause [17].

Network diagnostics based on traffic analysis can also use methods proposed for anomaly detection as some types of faults result in network communication anomalies.

Main contributions of our solution:

- automation of the tool Wireshark - Wireshark is a well-known protocol analyzer but lacks any task automation;
- the result is well understandable - the result contains steps which a real administrator would execute;
- easily extendable list of rules - the rules use Wireshark display filter language [18].

IV. Proposed System Architecture

We have built a proof-of-concept expert system to analyze network traffic. The system combines rule-based and inference-based approaches. We will not use a classifier-based approach [19], because it requires too much training data and only returns the root cause of the problem and not how it relates to the detected symptoms. Another benefit of the rule-based approach is that we can cover very specific situations for which getting training data could be very problematic.

We are focusing purely on passive methods because active methods are generating additional traffic into diagnosed networks (which is not acceptable for us) and also because this way, an administrator can perform an offline analysis on a computer not connected to the diagnosed network.

The proposed system processes the input data in several stages as shown in Figure 1. The first stage labeled as *Protocols Analyzer* filters and decodes input packets using an external tool. The second stage named *Events Finder* executes simple rules to identify events significant from the diagnostics point of view. In the third stage (*Tree Engine*), decision trees identify the possible problem cause and create a diagnostic output. All stages are easily extendable by the administrator who can add new rules and definitions.

Our proposed approach can also use different data sources (e.g., log files) as shown in Figure 1. *Events Finder* searches through data using analyzers specific to each data source. Additional analyzers could increase the diagnostic capability, however in our research, we are focusing only on network data, and we leave other possibilities for future research.

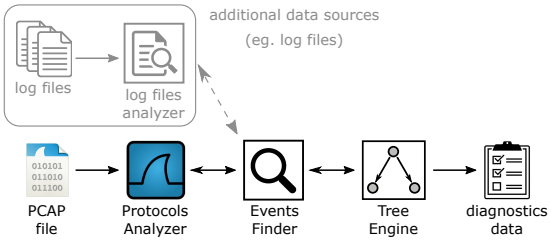

Figure 1. Top level architecture design of all the proposed system stages. The gray area represents optional extensions — additional data sources.

A. Protocols Analyzer

The first step in the processing pipeline is decoding captured network traffic in the PCAP format into a readable JSON format. We employ the tool *tshark*, which is a command

line version of the widely-used network protocol analyzer Wireshark. Because tshark follows the field naming convention used by Wireshark, we can use Wireshark Display Filter Expressions to select packet attributes. Tshark supports all packet dissectors available in Wireshark. Using tshark brings the following benefits:

- huge support of network protocols and when a new protocol is created, community can implement parsers very quickly and for free;
- adds tunneled, segmented and reassembled data support;
- tshark marks extracted data with the same names as displayed inside the Wireshark GUI. This allows a creation of easy-to-read API for diagnostics;

B. Events Finder

Events finder aims to identify events useful for network diagnosis (for example, a successful SMTP authentication event). An event rule consists of two parts: a list of packet filters and a list of assertions to express additional constraints. Both filter expressions and assertions use Wireshark's display filter language. Using this language, the expressions can be first tested in Wireshark before we use them in event finder rules.

The system evaluates the event rule as follows: (i) Each packet filter returns a list of packets matching the filter. (ii) Assertions are evaluated to select pairs of packets satisfying the constraints. A result has the form of a collection of pairs of packets, e.g., a rule that identifies DNS request-response pairs asserts that the transaction ID in both the request and response packets match. Assertion expressions use the display filter language extended with basic mathematical operations.

The event rules have the declarative specification written in YAML format. This format is described in subsection V-B. Rules are organized into modules. New modules can be easily added extending the rule database.

C. Tree Engine

The tree engine infers the possible error cause by evaluating a decision tree that contains expert knowledge about supported network protocols and services. Each node of the tree contains a diagnostic question. Questions refer to events identified by the *Event Finder*. Paths in the tree represent gathered knowledge and lead to the possible cause of the problem. Along the path, a diagnostic report is created to provide additional information for experienced users. The diagnostic report is produced in a human-readable format, as well as in a machine format useful for further processing or visualization.

The decision tree is comprised of the declarative specification of tree nodes enriched by Python code. Injection of Python code into the tree node definitions enables us to do complex knowledge processing. The idea is to keep the declarative part simple enough for most of the use-cases. The Python code is needed for specific use-cases, where a custom processing logic is necessary. The tree is defined using the YAML format rules, and subsection V-A describe its syntax.

V. Rule Specification

Diagnostic engine defines each protocol as a decision tree. The tree consists of nodes representing administrator questions, and edges representing answers to these questions. The edge

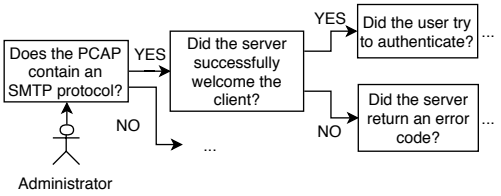

Figure 2. A simple illustration of a binary decision tree. Administrator diagnoses SMTP problem by checking questions in the predefined order.

can move the diagnostic process from one question to another or finish the process with the discovered result.

The questions simulate thinking of a real administrator. Typically, an administrator starts to search for certain network packet values and after the search for them is finished, the administrator searches for next values based on the result. In our solution, each question can only have two answers: success or fail. This yields a binary decision tree. Figure 2 shows an example of a small portion of the SMTP tree.

We need to convert the decision tree to a format understandable by our system. This conversion is split into two steps: 1) defining tree nodes (*Tree node rules*) and 2) defining conditions for choosing tree nodes (*Event condition rules*). The following subsection describes the syntax for both tree node rule and event condition rule. The reason why a node rule does not contain a condition code directly is that multiple rules would not be able to use the same condition code (reusability).

Conversion assigns a name to each node (*label_id*). We use the node names as labels for switching from one node to another. Each node has a condition (*condition_rule_id*), defined as an *Event condition rule*, used for choosing the next diagnostic step. Each rule can have one or none success and fail branch (*branch_code*). Branches contain executable Python code and the next node rule name. After the execution of the Python code, the analysis switches to the next node. Figure 3 shows the pseudocode for writing tree nodes.

```
1 label_id:
2   if (condition_rule_id):
3     success branch_code
4   else:
5     fail branch_code
```

Figure 3. Pseudocode for writing a tree node. Each node should have a unique id, condition, and branch codes.

A. Tree Node Rules

Each rule consists of an *event condition rule* name which should be executed, next states and blocks of Python code. The Python code can process packet data, make logical decisions and most importantly, generate diagnostic output. Instead of writing the whole output inside these rules, the rule contains only the name of the event. Each rule can switch to another protocol rule to diagnose problems across several protocols, e.g., if an SMTP communication is not detected, we will check if there are any ICMP unreachable messages, failed TCP connection attempts or incorrect DNS resolutions. Figure 4 shows an example of one rule defying the middle node from the tree in Figure 2.

```
id: smtp detected # name of the rule
query: welcome ok? # Events Finder rule
success:
  state: client welcomed # next state
  code: | # Python code follows
    event("client_welcomed")
fail:
  state: check error # next state
  code: | # Python code follows
    event("client_not_welcomed")
```

Figure 4. Simple Tree Engine rule showing what should be done if SMTP server welcomed the client or not.

B. Event Condition Rules

Rules in this section describe *how* the question is converted into packet lookup functions. Each rule may look for several independent packets, which are combined and checked if their relation fulfills the assert condition. Each question returns a list of tuples, where a tuple represents packets fulfilling the assert condition. Figure 5 shows an example of a simple rule for the question *Did the server successfully welcome the client?* Section *facts* looks for any hello commands and OK responses. The system puts founded packets which belong together into touples based on the *asserts* section.

```
id: welcome ok? # name of the rule
facts: # which packets we are looking for
  command: smtp.req.command in {"HELO"
           "EHLO"}
  reply: smtp.response.code == "250"
asserts: # packets relation constrain
 -command[tcp.stream]==reply[tcp.stream]
 -command[tcp.ack]==reply[tcp.seq]
```

Figure 5. Example of SMTP rule for checking if the server welcomed the client or not.

- SMTP: Connection detected
- SMTP: Server welcomed the client
- SMTP: Server is ready
- SMTP: Authentication 'gurpartap@patriots.in' - ok
- SMTP: The communication is not encrypted
- SMTP: No email has been sent
- SMTP: Transaction error code 552 - Requested mail actions aborted - Exceeded storage allocation
- SMTP: Empty email account storage (check SPAM folder) or increase the account quota.

Figure 6. An example of diagnostic output for an SMTP error. After an error 552 is detected and translated into human-readable error description, the system proposes a list of actions for fixing the error.

Before executing the diagnostic process, it is necessary to define event names from the *Tree engine* rules. A definition is just a simple dictionary which contains a severity and a description message. Part of the event description can be a pointer to another dictionary, which translates error codes to a human readable format. For example, instead of SMTP error code 552, the message "Requested mail actions aborted - Exceeded storage allocation" is displayed. After all rules and events are defined, it is possible to execute the diagnostic process. Figure 6 shows an example of one diagnostic output.

Table 1. Supported protocols and amount of rules and success, warning, error events which describe various protocol behavior situations.

Protocol	Node rules	Condition rules	Events		
			Success	Warning	Error
DHCP	25	23	10	9	4
DNS	12	12	8	2	6
FTP	24	10	17	5	6
ICMP	4	2	0	0	4
IMAP	15	8	7	0	11
POP	21	7	8	5	10
SIP	38	22	15	1	8
SLAAC	8	7	1	5	2
SMB	27	25	20	4	5
SMTP	17	13	10	5	9
SSL	1	1	1	0	1
TCP	11	11	0	8	3

VI. Conclusion

This paper presents a proposal of a system intended for troubleshooting network problems based on a passive network traffic analysis. The primary goal is to automate network diagnostics to help network administrators find causes of problems. The core of the presented approach is a multistage processing pipeline combining rule-based and inference-based methods. We have completed the implementation of a proof-of-concept system that we will use for preliminary evaluation and experiments.

We have implemented diagnostic rules for several application and service protocols. Table 1 shows the current list of supported protocols and their complexity in term of Node and Condition rule count, and their capabilities in term of Event count. After an evaluation of our solution by our partner — a monitoring vendor company, we have concluded, that the system must mark all reports which our tool may have incorrectly detected because of low-quality input data. For example, packet loss can drastically decrease the quality and accuracy of diagnostic results. In the current system, all reports from TCP flows with missing segments are marked as possibly incorrect.

Future work will focus on:

- evaluating the solution (accuracy and performance) and comparing the results with similar monitoring tools;
- analyzing each protocol's rules and based on used protocols and their field names create a filtering unit to reduce the amount of data processed by *Protocol Analyzer*;
- optimizing the performance. The current *Events Finder* combines all packets to check whether they are fulfilling the assert conditions or not. This all-to-all packet check has exponential time complexity ($2^{O(n)}$), which is unacceptable for large PCAP files. We want to optimize checking the asserts to decrease the complexity.

Acknowledgment

This work was supported by project "Network Diagnostics from Intercepted Communication" (2017-2019), no. TH02010186, funded by the Tech-nological Agency of the Czech Republic and by BUT project "ICT Tools, Methods and Technologies for Smart Cities" (2017-2019), no. FIT-S-17-3964.

REFERENCES

[1] R. Wang, D. Wu, Y. Li, X. Yu, Z. Hui, and K. Long, "Knights tour-based fast fault localization mechanism in mesh optical communication networks," Photonic Network Communications, vol. 23, no. 2, 2012, pp. 123–129.

[2] M. Solé, V. Muntés-Mulero, A. I. Rana, and G. Estrada, "Survey on models and techniques for root-cause analysis," arXiv preprint arXiv:1701.08546, 2017.

[3] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown, "A survey on network troubleshooting," Technical Report Stanford/TR12-HPNG-061012, Stanford University, Tech. Rep., 2012.

[4] M. Igorzata Steinder and A. S. Sethi, "A survey of fault localization techniques in computer networks," Science of computer programming, vol. 53, no. 2, 2004, pp. 165–194.

[5] C. Guo et al., "Pingmesh: A large-scale system for data center network latency measurement and analysis," in ACM SIGCOMM Computer Communication Review, vol. 45, no. 4. ACM, 2015, pp. 139–152.

[6] B. Agarwal et al., "Netprints: Diagnosing home network misconfigurations using shared knowledge," in Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, ser. NSDI'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 349–364.

[7] L. Lu, Z. Xu, W. Wang, and Y. Sun, "A new fault detection method for computer networks," Reliability Engineering & System Safety, vol. 114, 2013, pp. 45–51.

[8] S. Kandula et al., "Detailed diagnosis in enterprise networks," ACM SIGCOMM Computer Communication Review, vol. 39, no. 4, 2009, pp. 243–254.

[9] M. Luo, D. Zhang, G. Phua, L. Chen, and D. Wang, "An interactive rule based event management system for effective equipment troubleshooting," in IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society. IEEE, 2011, pp. 2329–2334.

[10] A. Mohamed, "Fault detection and identification in computer networks: A soft computing approach," Ph.D. dissertation, University of Waterloo, 2010.

[11] D. Brauckhoff, X. Dimitropoulos, A. Wagner, and K. Salamatian, "Anomaly extraction in backbone networks using association rules," in Proceedings of the 9th ACM SIGCOMM conference on Internet measurement. ACM, 2009, pp. 28–34.

[12] L. Benetazzo, C. Narduzzi, P. A. Pegoraro, and R. Tittoto, "Passive measurement tool for monitoring mobile packet network performances," IEEE transactions on instrumentation and measurement, vol. 55, no. 2, 2006, pp. 449–455.

[13] K.-H. Kim, H. Nam, J.-H. Park, and H. Schulzrinne, "Mot: a collaborative network troubleshooting platform for the internet of things," in Wireless Communications and Networking Conference (WCNC), 2014 IEEE. IEEE, 2014, pp. 3438–3443.

[14] T. Qiu, Z. Ge, D. Pei, J. Wang, and J. Xu, "What happened in my network: mining network events from router syslogs," in Proceedings of the 10th ACM SIGCOMM conference on Internet measurement. ACM, 2010, pp. 472–484.

[15] M. Vásquez-Bermúdez, J. Hidalgo, M. del Pilar Avilés-Vera, J. Sánchez-Cercado, and C. R. Antón-Cedeño, "Analysis of a network fault detection system to support decision making," in International Conference on Technologies and Innovation. Springer, 2017, pp. 72–83.

[16] S. Jamali and M. S. Garshasbi, "Fault localization algorithm in computer networks by employing a genetic algorithm," Journal of Experimental & Theoretical Artificial Intelligence, vol. 29, no. 1, 2017, pp. 157–174.

[17] E. S. Ali and M. Darwish, "Diagnosing network faults using bayesian and case-based reasoning techniques," in Computer Engineering & Systems, 2007. ICCES'07. International Conference on. IEEE, 2007, pp. 145–150.

[18] The Wireshark Wiki, "Displayfilters," [Online; accessed 20-April-2019]. [Online]. Available: https://wiki.wireshark.org/DisplayFilters

[19] C. Xu, H. Zhang, C. Huang, and D. Peng, "Study of fault diagnosis based on probabilistic neural network for turbine generator unit," in 2010 International Conference on Artificial Intelligence and Computational Intelligence, vol. 1. IEEE, 2010, pp. 275–279.

A.2 Using Rule-Based Decision Trees for Automatic Passive Diagnostics of the Network Problems

Authors: Holkovič Martin, Ing. (90%), Ryšavý Ondřej, doc. Ing., Ph.D. (10%)
Abstract: Network troubleshooting often requires a detailed analysis that may involve network packet capturing and a manual analysis using tools such as Wireshark. This is time-consuming and requires deep knowledge of communication protocols. Therefore this domain is a suitable candidate for the deployment of an expert system. In this paper, we consider a rule-based system integrating the expert knowledge that performs an automatic root cause analysis of network problems identifiable from network communications. The system is open, thus it is possible to add new rules as needed, e.g., for specific and recurring cases of a target environment. The rules are evaluated in a tree-based fashion, which enables us to collect additional information during the problem search to better explain the possible causes. We successfully deployed the tool as part of a commercial tool for network monitoring.
Keywords: Using Rule-Based Decision Trees for Automatic Passive Diagnostics of the Network Problems.
Published in: International Journal On Advances in Networks and Services
Impact Factor: 0.546
ISSN: 1942-2644

Using Network Traces to Generate Models for Automatic Network Application Protocols Diagnostics

Martin Holkovič[1], Ondřej Ryšavý[2] and Libor Polčák[2]
[1]*CESNET a.l.e., Zikova 1903/4, 160 00 Prague, Czech Republic*
[2]*Faculty of Information Technology, Brno University of Technology, Bozetechova 1/2, 612 66 Brno, Czech Republic*
holkovic@cesnet.cz, rysavy@fit.vutbr.cz and polcak@fit.vutbr.cz

Keywords: Network diagnostics, automatic diagnostics, protocol model from traces.

Abstract: Network diagnostics is a time-consuming activity that requires an administrator with good knowledge of network principles and technologies. Even if some network errors have been resolved in the past, the administrator must spend considerable time removing these errors when they reoccur. This article presents an automated tool to learn the expected behavior of network protocols and possible variations. The created model can be used to automate the diagnostic process. The model presents a finite automaton containing protocol behavior for different situations. Diagnostics of unknown communication is performed by checking the created model and searching for error states and their descriptions. We have also created a proof-of-concept tool that demonstrates the practical potential of this approach.

1 INTRODUCTION

Computer networks consist of a large number of devices and applications that communicate with each other. Due to the complexity of the network, errors occurred on a single device can negatively affect the network services and thus the user experience. There are various sources of error, such as misconfiguration, poor connectivity, hardware error, or even user misbehavior. End users are often unable to solve these problems and seek help from a network administrator. The administrator must diagnose the current situation, find the cause of the problem, and correct it to provide the service again.

The administrator diagnoses problems by checking communication and finding possible causes for these errors. Troubleshooting can be a rather complex activity requiring good technical knowledge of each network entity. Another complication is that the administrator often has to check the number of possible causes to find the true source of the problem, which requires some time. Network problems reappear even after the administrator detects and resolves these issues, such as repeating the same user error or when the application update on the server changes the expected behavior. All these problems make the diagnostic process a time-consuming and challenging activity that requires much administrator attention.

(Zeng et al., 2012) provides a short survey that shows that network diagnostics is time-consuming, and administrators wish to have a more sophisticated diagnostic tool available. Since each environment is different, the use of universal tools is difficult. It would be useful to have a tool that adapts to behavior on a particular network. The tool should learn the behavior of the network itself without the need to program or specify rules on the behavior of individual communicating applications and services.

Our goal is to develop a tool that automatically creates a protocol behavior model. We are not aiming at creating a general model for use in all networks, but the model should describe diagnosed network only. Instead of writing the model manually, the administrator provides examples (traces) of the protocol conversations in the form of PCAP files. The administrator provides two groups of files. The first group contains traces of normal behavior, while the second group consists of known, previously identified error traces. Based on these groups, the tool creates a protocol model. When the model is created, it can be used for detection and diagnosis of issues in the observed network communication. Once the model is created, additional traces may be used to improve the model gradually.

Our focus is on detecting application layer errors in enterprise networks. Thus, in the presented work, we do not consider errors occurred on other layers,

e.g., wireless communication (Samhat et al., 2007), routing errors (Dhamdhere et al., 2007), or performance issue on the network layer (Ming Luo, 2011). Because we are focusing on enterprise networks, we make some assumption on the availability of required source data. We expect that administrators using this approach have access to network traffic as the same administrators operate the network infrastructure, and it is possible to provide enough visibility to data communication. Even the communication outside the company's network is encrypted, the traffic between the company's servers and inside the network is many times unencrypted, or the data can be decrypted by providing server's private key or logging the symmetric session key [1]. Also, the source capture files have no or minimal packet loss. An administrator can recapture the traffic if necessary.

When designing the system, we assumed some practical considerations:

- no need to implement custom application protocol dissectors;
- application error diagnostics cannot be affected by lower protocols (e.g., version of IP protocol, data tunneling protocol);
- easily readable protocol model - the created model can be used for other activities too (e.g., security analysis).

To demonstrate the potential of our approach, we have created and evaluated a proof-of-concept implementation available as a command line tool.

The main contribution of this paper is a new automatic diagnostic method for error detection in network communication of commonly used application protocols. The method creates a protocol behavior model from packets traces that contain both correct and error communication patterns. The administrator can also use the created model for documentation purposes and as part of a more detailed analysis,e.g., performance or security analysis.

This paper is organized as follows: Section 2 describes existing work comparable to the presented approach. Section 3 overviews the system architecture. Section 4 provides details on the method, including algorithms used to create and use a protocol model. Section 5 presents the evaluation of the tool implementing the proposed system. Finally, Section 6 summarizes the paper and identifies possible future work.

[1] http://www.root9.net/2012/11/ssl-decryption-with-wireshark-private.html

2 RELATED WORK

Recently published survey paper (Tong et al., 2018) divides issues related to network systems as either *application-related* or *network-related* problems. Notable attention in troubleshooting of network applications was concentrated on networked multimedia systems, e.g., (Leaden, 2007), (Shiva Shankar and Malathi Latha, 2007), (Luo et al., 2007). Multimedia systems require that certain quality of service (QoS) is provided by the networking environment otherwise various types of issues can occur. Network issues comprise network reachability problems, congestion, excessive packet loss, link failures, security policy violation, and router misconfiguration.

Traditionally, network troubleshooting is a mostly manual process that uses several tools to gather and analyze relevant information. The ultimate tool for manual network traffic analysis and troubleshooting is Wireshark (Orzach, 2013). It is equipped with a rich set of protocol dissectors that enables to view details on the communication at different network layers. An administrator has to manually analyze the traffic and decide which communication is abnormal, possibly contributing to the observed problem. Though Wireshark offers advanced filtering mechanism, it lacks any automation (El Sheikh, 2018).

Network troubleshooting employs active, passive, or hybrid methods (Traverso et al., 2014). Active methods rely on the tools that generate probing packets to locate network issues (Anand and Akella, 2010). Specialized tools using generated diagnostic communication were also developed for testing network devices (Procházka et al., 2017). Contrary to active methods, the passive approach relies only on the observed information. Various data sources can be mined to obtain enough evidence to identify the problem. The collected information is then evaluated by the troubleshooting engine. The engine can use different techniques of fault localization.

Rule-based systems describe normal and abnormal states of the system. The set of rules is typically created by an expert and represents the domain knowledge for the target environment. Rule-based systems often do not directly learn from experience. They are also unable to deal with new previously unseen situations, and it is hard to maintain the represented knowledge consistently (łgorzata Steinder and Sethi, 2004).

Statistical and machine learning methods were considered for troubleshooting misconfigurations in home networks (Aggarwal et al., 2009) and diagnosis of failures in large Internet sites (Chen et al., 2004). *Tranalyzer* (Burschka and Dupasquier, 2017)

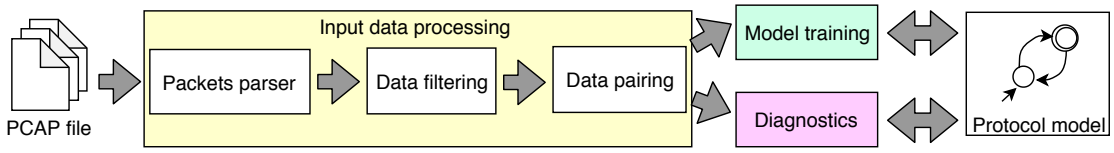

Figure 1: After the system processes the input PCAP files (the first yellow stage), it uses the data to create the protocol behavior model (the second green stage) or to diagnose an unknown protocol communication using the created protocol model (the-third purple stage).

is a flow-based traffic analyzer that performs traffic mining and statistical analysis enabling troubleshooting and anomaly detection for large-scale networks. Big-DAMA (Casas et al., 2016) is another framework for scalable online and offline data mining and machine learning supposed to monitor and characterize extremely large network traffic datasets.

Protocol analysis approach attempts to infer a model of normal communication from data samples. Often, the model has the form of a finite automaton representing the valid protocol communication. An automatic protocol reverse engineering that stores the communication patterns into regular expressions was suggested in (Xiao et al., 2009). Tool *ReverX* (Antunes et al., 2011) automatically infers a specification of a protocol from network traces and generates corresponding automaton. Recently, reverse engineering of protocol specification only from recorded network traffic was proposed to infer protocol message formats as well as certain field semantics for binary protocols (Lodi et al., 2018).

3 SYSTEM ARCHITECTURE

This section describes the architecture of the proposed system which learns from communication examples and diagnoses unknown communications. The system takes PCAP files as input data, where one PCAP file contains only one complete protocol communication. An administrator marks PCAP files as correct or faulty communication examples before model training. The administrator marks faulty PCAP files with error description and a hint on how to fix the problem. The system output is a model describing the protocol behavior and providing an interface for using this model for the diagnostic process. The diagnostic process takes a PCAP file with unknown communication and checks whether this communication contains an error and if yes, returns a list of possible errors and fixes.

The architecture, shown in Figure 1, consists of multiple components, each implementing a stage in the processing pipeline. The processing is staged as follows:

- **Input data processing** - Preprocessing is responsible for converting PCAP files into a format suitable for the next stages. Within this stage, the input packets are decoded using protocol parser. Next, the filter is applied to select only relevant packets. Finally, the packets are grouped to pair request to their corresponding responses.
- **Model training** - The training processes several PCAP files and creates a model characterizing the behavior of the analyzed protocol. The output of this phase is a protocol model.
- **Diagnostics** - In the diagnostic component, an unknown communication is analyzed and compared to available protocol models. The result is a report listing detected errors and possible hints on how to correct them.

In the rest of the section, the individual components are described in detail. Illustrative examples are provided for the sake of better understanding.

3.1 Input Data Processing

This stage works directly with PCAP files provided by the administrator. Each file is parsed by *TShark* [2] which exports decoded packets to JSON format. The system further processes the JSON data by filtering irrelevant records and pairs request packets with their replies. The output of this stage is a list of tuples representing atomic transactions.

3.1.1 Packets parser

Instead of writing our packet decoders, we use the existing implementation provided by *TShark*. TShark is the console version of the well-known Wireshark protocol analyzer which supports many network protocols and can also analyze tunneled and fragmented packets. In the case the Wireshark does not support some protocol, e.g., proprietary, it is possible to use a tool which generates dissectors from XML files (Golden and Coffey, 2015). The system converts each input PCAP file into the JSON format (TShark supports multiple formats). The JSON

[2] https://www.wireshark.org/docs/man-pages/tshark.html

```
...
"eth": {
  "eth.dst": "f0:79:59:72:7c:30",
  "eth.type": "0x00000800",
  ...
},
...
"dns": {
  "dns.id": "0x00007956",
  "dns.flags.response": "0",
  "dns.flags.opcode": "0",
  "dns.qry.name": "mail.patriots.in",
  ...
},
...
```

Figure 2: Excerpt from the TShark output into JSON format. The JSON represents POP3 packet values from all network protocols in a key-value data format.

format represents data as a key-value structure (see Figure 2), where the key is the name of the protocol field according to Wireshark definition [3], e.g., *pop.request.command*.

3.1.2 Data filtering

The JSON format from the TShark output is still protocol dependent because the field names are protocol-dependent, and we have to know which key names each protocol uses. The system converts the data into a more generic format to make the next processing protocol independent. We have found out, that most of the application protocols use a request-reply communication pattern. The system filters requests and replies from each protocol and removes the rest of the data. Even though the protocols use the same communication pattern, they use a different naming convention to mark the reply and response values (see Table 1).

The problem is how to find the requests and replies in the JSON data. In our solution, we have created a database of protocols and their processed field names. In the case the protocol is not yet in the database, we require the administrator to add these two field names to the database. The administrator can get the field names from the Wireshark tool easily by clicking on the appropriate protocol field.

Messages can also contain additional information and parameters, e.g., server welcome message. The system also removes this additional information to allow generalization of otherwise different messages during the protocol model creation. For example, the welcome server message often contains

[3] https://www.wireshark.org/docs/dfref/

Table 1: Several application protocols with their request and reply field names with example values. The system takes only data from these fields and drops the rest.

Name	Type	Field name	E.g.
SMTP	Request	smtp.req.command	MAIL
	Reply	smtp.response.code	354
FTP	Request	ftp.request.command	RETR
	Reply	ftp.response.code	150
POP	Request	pop.request.command	STAT
	Reply	pop.response.indicator	+OK
DNS	Request	dns.flags.opcode	0
	Reply	dns.flags.rcode	0

the current date and time, which is always different, and these different messages would create a lot of unrepeatable protocol states. The Figure 3 shows the resulting format from the *Data filtering* step.

Unfortunately, TShark marks some unpredictable data (e.g., authentication data) in some protocols as regular requests and does not clearly distinguish it. These values are a problem in later processing because these unpredictable values create ungeneralizable states during the protocol model learning phase. In our tests, we have observed that regular requests have a maximal length of 6 characters, and unpredictable requests have a much longer length. Based on our finding, the system drops all requests that are longer than six characters, and if necessary, the network administrator can change this value. Distinguishing of these unpredictable requests also for other protocols should be more focused in future research.

3.1.3 Data pairing

To avoid pairing requests and replies during the protocol model learning process, the system pairs each reply to its request in the data processing stage. After this pairing process, the system represents each protocol with a list of pairs, each pair containing one request and one reply. This pairing also simplifies the model learning process because some protocols use the same reply value for multiple commands (e.g., POP3 reply *"+OK"* for all correct replies) and the model could improperly merge several independent reply values into one state (e.g., all POP3 success replies jumps into one state).

The pairing algorithm iteratively takes requests one by one and chooses the first reply it finds. If no reply follows, the system pairs the request with a *"NONE"* reply. In some protocols, the server sends a reply to the client immediately after they connect to the server. E.g., the POP server informs the clients if it can process their requests or not. We have solved this problem by pairing these replies with the empty request *"NONE"*. The result of the pairing process

is a sequence of pairs, where each pair consists of one request and one reply. The Figure 3 shows an example of this pairing process.

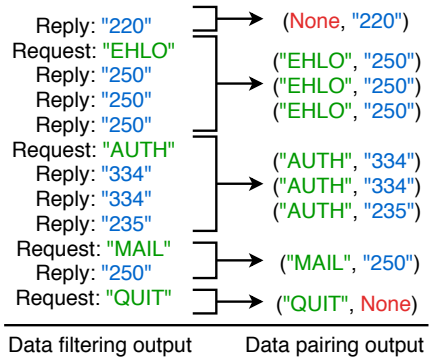

Figure 3: Example of an SMTP communication in which the client authenticates, sends an email and quits the communication. The left part of the example shows output from the *Data filtering* stage containing a list of requests and replies in the protocol-independent format. The right part shows a sequence of paired queries with replies, which are the output of the *Data pairing* stage. The system pairs one request and one reply with the special *None* value.

3.2 Model Training

After the *Input Data Processing* stage transformed input PCAP files into a list of request-response pairs, the *Model Training* phase creates a model of the protocol. The model has the form of a finite state machine describing the behavior of the protocol. The system creates the model from provided communication traces. For example, for POP3 protocol, we can consider regular communication traces that represent typical operations, e.g., the client is checking a mail-box on the server, downloading a message from the server or deleting a message on the server. The model is first created for regular communication and later extended with error behavior.

Learning from traces with expected behavior

The model creation process begins by learning the protocol behavior from input data representing regular communication. The result of this training phase is a description of the protocol that represents a subset of correct behavior. The model is created from a collection of individual communication traces. When a new trace is to be added, the tool identifies the longest prefix of the trace that is accepted by the current model. The remaining of the trace is then used to enrich the model. The Figure 4 shows a simple example of creating a model from two correct communication traces (drawn in black ink).

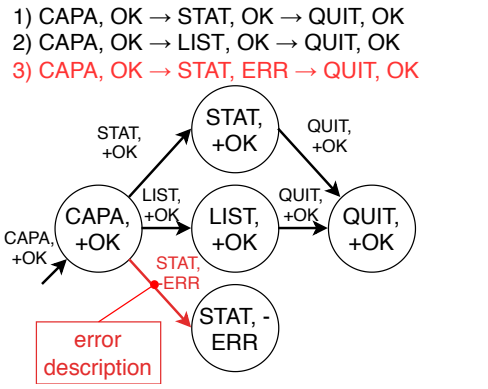

Figure 4: An example of communication traces and the corresponding protocol model. The first two sequences represent correct communication, while the third sequence is communication with an error.

Learning the errors

After the system learns the protocol from regular communication, the model can be extended with error traces. In Figure 4, red arrow stands for a single error transition in the model that corresponds to the added error trace. The system expects that the administrator prepares that error trace as the result of previous (manual) troubleshooting activities. The administrator should also provide error description and information about how to fix the error.

When extending the model with error traces, the procedure is similar to when processing correct traces. Automaton attempts to consume as long prefix of input trace as possible ending in state s. The following cases are possible:

- *Remaining input trace is not empty*: The system creates a new state s' and links it with from state s. It marks the new state as an error state and labels it with a provided error description.
- *Remaining input trace is empty*:
  - State s is error state: The system adds the new error description to existing labeling of an existing state s.
  - State s is correct state: The system marks the state as *possible error* and adds the error description.

When extending the automaton with error traces, it is possible that previously correct state is changed to a possible error state. For consistent application protocols, this ambiguity is usually caused by the abstraction made when describing application protocol behavior.

3.3 Diagnostics

After the system creates a behavioral model that is extended by error states, it is possible to use the model to diagnose unknown communication tracks. The system runs diagnostics by processing a PCAP file in the same way as in the learning process and checks the request/response sequence against the automaton. Diagnostics distinguishes between these classes:

- **Normal:** the automaton accepts the entire input trace and ends in the correct state.
- **Error:** the automaton accepts the entire input trace and ends in the error state.
- **Possible error:** the automaton accepts the entire input trace and ends in the possible error state. In this case, the system cannot distinguish if the communication is correct or not. Therefore, the system reports an error description from the state and leaves the final decision on the user.
- **Unknown:** the automaton does not accept entire the input trace, which may indicate that the trace represents a behavior not fully recognized by the underlying automaton.

If the diagnostic process detects an unknown error or result is not expected, the administrator must manually analyze the PCAP file. After the administrator decides whether the file contains an error or not, the administrator should assign a file to a particular group of files (correct or error) and repeat the learning process. This re-learning process increases the model's ability, and next time the system sees the same situation, it reports the correct result. By gradually expanding, the model covers most of the possible options.

4 ALGORITHMS

This section provides algorithms for (i) creating a model from normal traces, (ii) updating the model from error traces and (iii) evaluating a trace if it contains an error. All three presented algorithms work with a model that uses a deterministic finite automaton (DFA) as its underlying representation.

The protocol behavior is an automaton $(Q, \Sigma, \delta, q_0, F)$. The set of states Q is represented by all query/response pairs identified for the modeled application protocol. As $Q \subseteq \Sigma$, the transition relation $\delta : Q \times \Sigma \rightarrow Q$ is restricted as follows:

$$\delta \subseteq \{((q_s, r_s), (q_i, r_i), (q_i, r_i)) | (q_s, r_s), (q_i, r_i) \in Q\}$$

Each state can be a finite state because the input of the respective input is an indication of the state reached and a list of error descriptions obtained when processing the input data.

4.1 Adding Correct Traces

Algorithm 1 takes the current model (input variable DFA) and adds missing transitions and states based on the input sequence (input variable P). The algorithm starts with the init state and saves it into the *previous_state* variable. The *previous_state* variable is used to create a transition from one state to the next. In each loop of the while loop section, the algorithm assigns the next pair into the *current_state* variable until there is no next pair in the input. From the *previous_state* and the *current_state*, the transition variable is created, and the system checks if the DFA contains this transition. If the DFA does not contain the transition, the transition is added to the DFA. Before continuing with the next loop, the *current_state* variable is assigned to the *previous_state* variable. The updated model will be used as the input for the next unprocessed input sequence. After processing all the input sequences, which represent normal behavior, the resulting automaton is a model of normal behavior.

Algorithm 1 Updating model from the correct traces

Inputs: P = sequence of query-reply pairs
DFA = set of the transitions
Output: DFA = set of the transitions
Previous_state = *init_state*
while *not at end of input P* **do**
  Current_state = get next pair from *P*
  Transition = *Previous_state* $\rightarrow$ *Current_state*
  if *DFA does not contain Transition* **then**
    add *Transition* to DFA
  Previous_state = *Current_state*
end
return DFA

4.2 Adding Error Traces

The Algorithm 2 has one more input (*Error*), which is a text string describing a user-defined error. The start of the algorithm is the same as in the previous case. The difference is in testing whether the automaton contains the transition specified in the input sequence. If so, the system checks to see if the saved transition also contains errors. In this case, the algorithm updates the error list by adding a new error. Otherwise, the algorithm continues to process the input string to find a suitable place to indicate the error. If the transition does not exist, i is created and marked with the specified error.

Algorithm 2 Extending the model with error traces

Inputs: P = sequence of query-reply pairs
DFA = set of transitions
Error = description of the error
Output: DFA = set of transitions
Previous_state = *init_state*
while *not at end of input P* **do**
  Current_state = get next pair from *P*
  Transition = *Previous_state* → *Current_state*
  if *DFA contains Transition* **then**
    if *Transition contains error* **then**
      append *Error* to *Transition* in DFA
      return DFA
    else
      Previous_state = *Current_state*
  else
    add transition *Transition* to DFA
    mark *Transition* in DFA with *Error*
    return DFA
end
return DFA

Algorithm 3 Checking an unknown trace

Inputs: P = sequence of query-reply pairs
DFA = set of transitions
Output: Errors = one or more error descriptions
Previous_state = *init_state*
while *not at end of input P* **do**
  Current_state = get next pair from *P*
  Transition = *Previous_state* → *Current_state*
  if *DFA contains Transition* **then**
    if *Transition contains error* **then**
      return *Errors* from *Transition*
    else
      Previous_state = *Current_state*
  else
    return "unknown error"
end
return "no error detected"

4.3 Testing Unknown Trace

The Algorithm 3 uses previously created automaton (DFA variable) to check the input sequence P. According to the input sequence, the algorithm traverses the automaton and collects the errors listed in the transitions taken. If the required transition was not found, the algorithm returns an error. In this case, it is up to the user to analyze the situation and possibly extend the automaton for this input.

5 EVALUATION

We have implemented a proof-of-concept tool which implements the Algorithm 1, 2, and 3 specified in the previous section. In this section, we provide the evaluation of our proof-of-concept tool to demonstrate that the proposed solution is suitable for diagnosing application protocols. Another goal of the evaluation is to show how the created model changes by adding new input data to the model. We have chosen four application protocols with different behavioral patterns for evaluation.

5.1 Reference Set Preparation

Our algorithms create the automata states and transitions based on the sequence of pairs. The implication is that repeating the same input sequence does not modify the learned behavior model. Therefore, it is not important to provide a huge amount of input files (traces) but to provide unique traces (sequences of query-reply pairs). We created our reference datasets by capturing data from the network, removing unrelated communications, and calculating the hash value for each trace to avoid duplicate patterns. Instead of a correlation between the amount of protocols in the network and the amount of saved traces, the amount of files correlates with the complexity of the analyzed protocol. For example, hundreds of DNS query-reply traces captured from the network can be represented by the same sequence (dns_query, dns_reply).

After capturing the communication, all the traces were manually checked and divided into two groups: (i) traces representing normal behavior and (ii) traces containing some error. In case the trace contains an error, we also identified the error and added the corresponding description to the trace. We split both groups of traces (with and without error) into the training set and the testing set.

It is also important to notice that the tool uses these traces to create a model for one specific (or several) network configuration and not for all possible configurations. Focus on a single configuration results in a smaller set of unique traces and smaller created models. This focus allows an administrator to detect situations which may be correct for some network, but it is not correct for a diagnosed network, e.g., missing authentication.

5.2 Model Creation

We have chosen the following four request-reply application protocols with different complexity for evaluation:

Table 2: For each protocol, the amount of total and training traces is shown. These traces are separated into successful (without error) and failed (with error) groups. The training traces are used to create two models, the first without errors and the second with errors. The states and transitions columns indicate the complexity of the created models.

Protocol	Total traces		Training traces		Model without error states		Model with error states	
	Successful	Failed	Successful	Failed	States	Transitions	States	Transitions
DNS	16	8	10	6	18	28	21	34
SMTP	8	4	6	3	11	18	14	21
POP	24	9	18	7	16	44	19	49
FTP	106	20	88	14	33	126	39	137

- **DNS**: Simple stateless protocol with simple communication pattern - domain name query (type A, AAAA, MX, ...) and reply (no error, no such name, ...).
- **SMTP**: Simple state protocol in which the client has to authenticate, specify email sender and recipients, and transfer the email message. The protocol has a large predefined set of reply codes resulting in many possible states in DFA created by Algorithm 1 and 2.
- **POP**: In comparison with SMTP, from one point of view, the protocol is more complicated because it allows clients to do more actions with email messages (e.g., download, delete). However, the POP protocol replies only with two possible replies (+OK, -ERR), which reduce the number of possible states.
- **FTP**: It is stateful protocol usually requires authentication, then allows the client to do multiple actions with files and directories, and also the protocol defines many reply codes.

The proof-of-concept tool took input data of selected application protocols and created models of the behavior without errors and a model with errors. The Table 2 shows the distribution of the input data into a group of correct training traces and a group of traces with errors. Remaining traces will be later used for testing the model. The right part of the table shows the complexity of the generated models in the format of states and transitions count.

Based on the statistics of models, we have made the following conclusions:

- transitions count represents the complexity of the model better than the state's count;
- there is no direct correlation between the complexity of the protocol and the complexity of the learned model. As can be seen with protocols DNS and SMTP, even though the model SMTP is more complicated than DNS protocol, there were about 50% fewer unique traces resulting in a model with 21 transitions, while the DNS model consists of 34 transitions. The reason for this situation is that one DNS connection can contain more than one query-reply and because the protocol is stateless, any query-reply can follow the previous query-reply value.

Figure 5 shows four charts of four protocols that show the complexity of the models in terms of the number of states, transitions, and testing traces with error results. Each chart consists of two parts. The first part marked as *training from correct traces* creates the model only from traces without errors. To check the correctness of the model, we used testing traces with and without errors. The second part *learning the errors* takes the model created from all the successful traces and extends it with training traces with known errors. To mark the testing trace without error as a correct result, the model has to return that the trace is without error. Testing traces with an error are marked as correct when an error was detected, and the model found the correct error description.

The charts in Figure 5 shows the progress of changing the model size when new traces are added to the model. We have created these values from 25 tests, and the charts show a range of the values from these tests with their median. Each test began by randomizing the order of the trace files, resulting in a different trace order in each test. Based on the deviation of values from the median, we can see that during the learning process, the model is dependent on the order of the traces. However, after we have added all the traces, the created model has the same amount of states and transitions (zero deviation). The zero deviation can be seen at the end of training from correct trace states and also at the end of learning the error states. We have also used a diff tool to compare the final models between themselves to confirm that all the models were the same, and the final model does not depend on the order of the input traces.

Figure 5 shows that by adding new traces, the size of the model is increasing. With the increasing size of the model, the model is more accurate, and

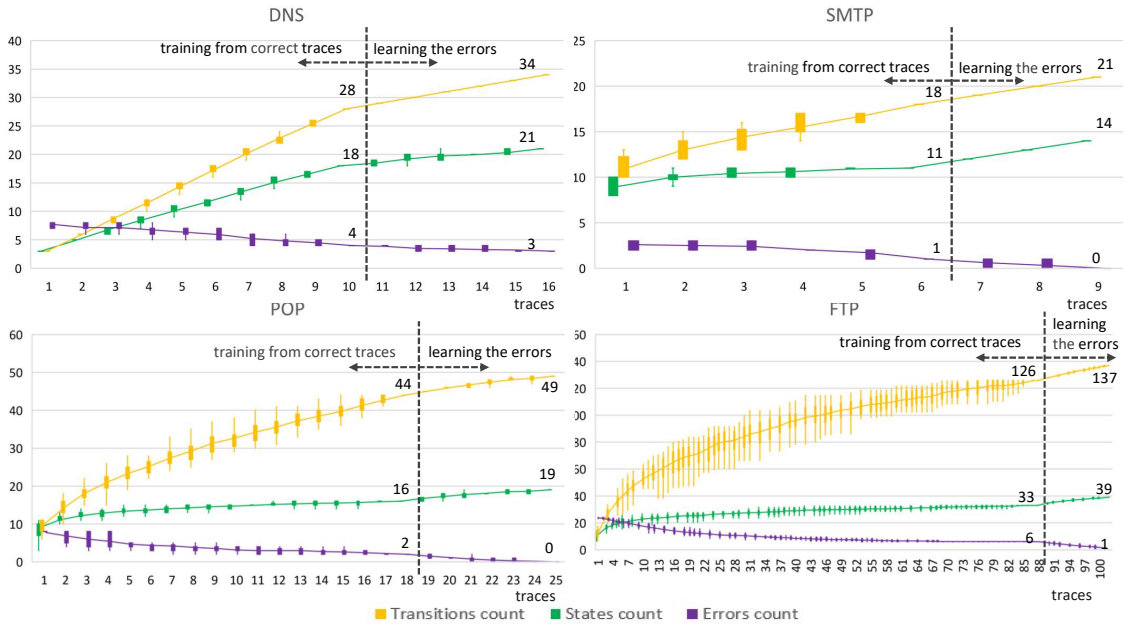

Figure 5: The figure shows the count of transitions, states, and errors in the four analyzed protocols. An error is an incorrect diagnostic result. The values are extracted from 25 random tests, and the median of their values is represented by interconnection lines. The learning process is split into two parts: i) training the model only from traces without any error and after all correct traces have been learned, in section ii) model is extended with the knowledge of known errors.

the amount of diagnostic error results decreases. However, after some amount of traces, the model expansion will slow down until it stops after the tool has observed all valid traces. Stopping the expansion may seem like the point when the model is fully trained, however from our experience, it is not possible to determine when the model is fully learned or at least learned from X%. Even if the model does not grow for a long time, it can suddenly expand by processing a new trace (new extensions, programs with specific behavior, program updates).

Another way of specifying how much percent the model is trained is by calculating all possible transitions. The calculation is $(requests\_count * replies\_count)^2$. Of course, many combinations of requests and replies would not make any sense, but the algorithm can never be sure which combinations are valid and which are not. The problem with counting all possible combinations is that without predefined knowledge of diagnosed protocol the tool can never be sure if all possible requests and replies (no matter the combinations) have already be seen or not.

5.3 Evaluation of Test Traces

Table 3 shows the amount of successful and failed testing traces; the right part of Table 3 shows testing results for these data. All tests check whether:

1. a successful trace is marked as correct (TN);
2. a failed trace is detected as an error trace with correct error description (TP);
3. a failed trace is marked as correct (FN);
4. a successful trace is detected as an error or failed trace is detected as an error but with an incorrect error description (FP);
5. true/false (T/F) ratios which are calculated as $(TN+TP)/(FN+FP)$. T/F ratios represents how many traces the model diagnosed correctly.

As the columns T/F ratio in Table 3 shows, most of the testing data was diagnosed correctly. We have analyzed the incorrect results and made the following conclusions:

- **DNS**: False positive - One application has made a connection with the DNS server and keeps the connection up for a long time. Over time several queries were transferred. Even though the model contains these queries, the order in which they came is new to the model. The model returned an error result even when the communication ended correctly. An incomplete model causes this misbehavior. To correctly diagnose all query combinations, the model has to be created from more unique training traces.
- **DNS**: False positive - The model received a new SOA update query. Even if the communication

Table 3: The created models have been tested by using testing traces, which are split into successful (without error) and failed (with error) groups. The correct results are shown in the true negative and true positive columns. The columns false positive and false negative on the other side contain the number of wrong test results. The ratio of correct results is calculated as a true/false ratio. This ratio represents how many testing traces were diagnosed correctly.

Protocol	Testing traces		Testing against model without error states					Testing against model with error states				
	Successful	Failed	TN	TP	FN	FP	T/F ratio	TN	TP	FN	FP	T/F ratio
DNS	6	2	4	2	0	2	75 %	4	1	1	2	63 %
SMTP	2	1	2	1	0	0	100 %	2	1	0	0	100 %
POP	6	2	6	2	0	0	100 %	6	2	0	0	100 %
FTP	18	6	18	6	0	0	100 %	18	5	1	0	96 %

TN - true negative, TP - true positive, FN - false negative, FP - false positive, T/F ratio - true/false ratio

did not contain the error by itself, it is an indication of a possible anomaly in the network. Therefore, we consider this as the expected behavior.

- **DNS**: False negative - The situation was the same as with the first DNS False positive mistake - the order of packets was unexpected. Unexpected order resulted in an unknown error instead of an already learned error.
- **FTP**: False negative - The client sent a PASS command before the USER command. This resulted in an unexpected order of commands, and the model detected an unknown error. We are not sure how this situation has happened, but because it is nonstandard behavior, we are interpreting this as an anomaly. Hence, the proof-of-concept tool provided the expected outcome.

All the incorrect results are related to the incomplete model. In the real application, it is almost impossible to create a complete model even with many input data. In the stateless protocols (like DNS), it is necessary to capture traces with all combinations of query-reply states. For example, if the protocol defines 10 types of queries, 3 types of replies, the total amount of possible transitions is $(10 * 3)^2 = 900$. Another challenge is a protocol which defines many error reply codes. To create a complete model, all error codes in all possible states need to be learned from the traces.

We have created the tested tool as a prototype in Python language. We have not aimed at testing the performance, but to get at least an idea of how usable our solution is, we gathered basic time statistics. The processing time of converting one PCAP file (one trace) into a sequence of query-replies and adding it to the model took on average 0.4s. This time had only small deviations because most of the time took initialization of the TShark. The total amount of time required to learn a model depends on the amount of PCAPs. The average time required to create a model from 100 PCAPs was 30 seconds.

6 CONCLUSIONS

This paper suggested a method for automatic error diagnostics in network application protocols by creating models for these applications. There are two use-cases for when administrators should use this approach: (i) if an administrator is experienced, the administrator can learn the model to speed-up the diagnostic process; (ii) if an administrator is inexperienced, the administrator can use the model created by an experienced administrator to diagnose the network.

The already existing diagnostic solutions do not have any automation capabilities, require an administrator to create rules describing the normal and error states or the automatically created protocol models are not used for diagnostic purposes.

Our method uses network traces prepared by administrators to create a model representing protocol behavior. The administrator has to separate the correct from the error traces and annotate the error ones. The model is created based on the query-response sequences extracted from the analyzed protocol. For this reason, the model is applicable only for a protocol with a query-reply pattern. The model represents the correct and incorrect protocol behavior, which is used for unknown network trace diagnostics.

The main benefit of having an own trained model is that the model will represent the protocol in a specific configuration and will not accept situations which may be valid only for other networks. The administrator can use the model to speed up the work by automating unknown communication diagnostics. Our solution uses the well-known tool TShark, which supports many network protocols and allows us to use the Wireshark display language to mark the data.

We have implemented the proposed method as a proof-of-concept tool [4] to demonstrate its capabilities. The tool has been tested on four application protocols that exhibit different behavior. Experiments

[4] https://github.com/marhoSVK/semiauto-diagnostics

have shown that it is not easy to determine when a model is sufficiently taught. Even knowing the protocol complexity is not a reliable indication. Because even the simplest protocol we tested needed a larger model than a more complex protocol. Although the model seldom covers all possible situations, it is useful for administrators to diagnose repetitive and typical protocol behavior and find possible errors.

Future work will focus on: (i) finding other automation cases for the created protocol model ; (ii) designing and implementing additional communication modeling algorithms to support other useful communication features ; (iii) a study on possibility of combining different models from multiple algorithms into one complex model; (iv) integrating timing information into DFA edges.

ACKNOWLEDGEMENTS

This work was supported by project "Network Diagnostics from Intercepted Communication" (2017-2019), no. TH02010186, funded by the Technological Agency of the Czech Republic and by BUT project "ICT Tools, Methods and Technologies for Smart Cities" (2017-2019), no. FIT-S-17-3964.

REFERENCES

Aggarwal, B., Bhagwan, R., Das, T., Eswaran, S., Padmanabhan, V. N., and Voelker, G. M. (2009). NetPrints: Diagnosing home network misconfigurations using shared knowledge. *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, Di(July):349–364.

Anand, A. and Akella, A. (2010). {NetReplay}: a new network primitive. *ACM SIGMETRICS Performance Evaluation Review*.

Antunes, J., Neves, N., and Verissimo, P. (2011). Reverx: Reverse engineering of protocols. Technical Report 2011-01, Department of Informatics, School of Sciences, University of Lisbon.

Burschka, S. and Dupasquier, B. (2017). Tranalyzer: Versatile high performance network traffic analyser. In *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016*.

Casas, P., Zseby, T., and Mellia, M. (2016). Big-DAMA: Big Data Analytics for Network Traffic Monitoring and Analysis. *Proceedings of the 2016 Workshop on Fostering Latin-American Research in Data Communication Networks (ACM LANCOMM'16)*.

Chen, M., Zheng, A., Lloyd, J., Jordan, M., and Brewer, E. (2004). Failure diagnosis using decision trees. *International Conference on Autonomic Computing, 2004. Proceedings.*, pages 36–43.

Dhamdhere, A., Teixeira, R., Dovrolis, C., and Diot, C. (2007). NetDiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data. *Proceedings of the 2007 ACM CoNEXT*.

El Sheikh, A. Y. (2018). Evaluation of the capabilities of wireshark as network intrusion system. *Journal of Global Research in Computer Science*, 9(8):01–08.

Golden, E. and Coffey, J. W. (2015). A tool to automate generation of wireshark dissectors for a proprietary communication protocol. *The 6th International Conference on Complexity, Informatics and Cybernetics, IMCIC 2015*.

Leaden, S. (2007). The Art Of VOIP Troubleshooting. *Business Communications Review*, 37(2):40–44.

łgorzata Steinder, M. and Sethi, A. S. (2004). A survey of fault localization techniques in computer networks. *Science of computer programming*, 53(2):165–194.

Lodi, G., Buttyon, L., and Holczer, T. (2018). Message Format and Field Semantics Inference for Binary Protocols Using Recorded Network Traffic. In *2018 26th International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2018*.

Luo, C., Sun, J., and Xiong, H. (2007). Monitoring and troubleshooting in operational IP-TV system. *IEEE Transactions on Broadcasting*, 53(3):711–718.

Ming Luo, Danhong Zhang, G. P. L. C. (2011). An interactive rule based event management system for effective equipment troubleshooting. *Proceedings of the IEEE Conference on Decision and Control*, 8(3):2329–2334.

Orzach, Y. (2013). *Network Analysis Using Wireshark Cookbook*. Packt Publishing Ltd.

Procházka, M., Macko, D., and Jelemenská, K. (2017). IP Networks Diagnostic Communication Generator. In *Emerging eLearning Technologies and Applications (ICETA)*, pages 1–6.

Samhat, A., Skehill, R., and Altman, Z. (2007). Automated troubleshooting in WLAN networks. In *2007 16th IST Mobile and Wireless Communications Summit*.

Shiva Shankar, J. and Malathi Latha, M. (2007). Troubleshooting SIP environments. In *10th IFIP/IEEE International Symposium on Integrated Network Management 2007, IM '07*.

Tong, V., Tran, H. A., Souihi, S., and Mellouk, A. (2018). Network troubleshooting: Survey, Taxonomy and Challenges. *2018 International Conference on Smart Communications in Network Technologies, SaCoNeT 2018*, pages 165–170.

Traverso, S., Tego, E., Kowallik, E., Raffaglio, S., Fregosi, A., Mellia, M., and Matera, F. (2014). Exploiting hybrid measurements for network troubleshooting. In *2014 16th International Telecommunications Network Strategy and Planning Symposium, Networks 2014*.

Xiao, M. M., Yu, S. Z., and Wang, Y. (2009). Automatic network protocol automaton extraction. In *NSS 2009 - Network and System Security*.

Zeng, H., Kazemian, P., Varghese, G., and McKeown, N. (2012). A survey on network troubleshooting. *Technical Report Stanford/TR12-HPNG-061012, Stanford University, Tech. Rep.*

A.3 Automating Network Security Analysis at Packet-level by using Rule-based Engine

Authors: Holkovič Martin, Ing. (85%), Ryšavý Ondřej, doc. Ing., Ph.D. (13%), Dudek Jindřich, Ing. (2%)

Abstract: When a network incident is detected, a network administrator has to manually verify the incident and provide a solution to stop the incident from continuing and prevent similar incidents in the future. The network analysis is a time-consuming and labor-intensive activity which requires good network knowledge. Creating a solution which automates the administrator's work can dramatically speed up the analysis process and can make the whole process easier for less experienced administrators. In this paper, we describe a method that uses a predefined set of rules to identify incident patterns. Though this principle is used by many security tools, the new aspect is that the presented approach uses the Wireshark tool which is well known among the administrators, and it is expressive enough to specify complex relations among source data thus being able to detect quite sophisticated attacks. The created rule's format uses the same language as the Wireshark filters.

Keywords: Network security, network monitoring, anomaly detection, threat detection, network forensics.
Published in: 6th Conference on the Engineering of Computer Based Systems (ECBS 2019), Bucharest, Romania
Conference rating: B (Core), B1 (Qualis)
ISBN: 978-1-4503-7636-5

Automating Network Security Analysis at Packet-level by using Rule-based Engine

Martin Holkovič
Brno University of Technology
Brno, Czech Republic
iholkovic@fit.vutbr.cz

Ondřej Ryšavý
Brno University of Technology
Brno, Czech Republic
rysavy@fit.vutbr.cz

Jindřich Dudek
Brno University of Technology
Brno, Czech Republic
xdudek04@fit.vutbr.cz

ABSTRACT

When a network incident is detected, a network administrator has to manually verify the incident and provide a solution to stop the incident from continuing and prevent similar incidents in the future. The network analysis is a time-consuming and labor-intensive activity which requires good network knowledge. Creating a solution which automates the administrator's work can dramatically speed up the analysis process and can make the whole process easier for less experienced administrators. In this paper, we describe a method that uses a predefined set of rules to identify incident patterns. Though this principle is used by many security tools, the new aspect is that the presented approach uses the Wireshark tool which is well known among the administrators, and it is expressive enough to specify complex relations among source data thus being able to detect quite sophisticated attacks. The created rule's format uses the same language as the Wireshark filters.

CCS CONCEPTS

• **Networks** → **Error detection and error correction**; **Network monitoring**; *Network security*; • **Security and privacy** → *Intrusion detection systems.*

KEYWORDS

Network security, network monitoring, anomaly detection, threat detection, network forensics

ACM Reference Format:
Martin Holkovič, Ondřej Ryšavý, and Jindřich Dudek. 2019. Automating Network Security Analysis at Packet-level by using Rule-based Engine. In *6th Conference on the Engineering of Computer Based Systems (ECBS '19), September 2–3, 2019, Bucharest, Romania.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3352700.3352714

1 INTRODUCTION

Nowadays, one of the most important responsibilities of a network administrator is to secure a computer network against cyber threats. A threat can have a lot of different formats: brute-force login attempt, DDoS attack, phishing attack, data breach, and many others. Most of the protective actions are done proactively to prevent threats realization, but despite all the effort and implemented measures, some incident can occur in the network anyway. Then a network administrator has to identify, locate and stop incidents inside the network [23]. In this paper, we will use an example of a user who cannot access the Internet. The administrator has to investigate this problem and fix the user's Internet connection.

A very common way of network traffic analysis is using some network packet analyzer, e.g., *Wireshark* [18]. The analyzer process captured network traffic (PCAP files) and decodes individual packets. The administrator analyzes available information, checks the transferred data and compares it with expected values (e.g., from RFC standards). This manual process is time-consuming and requires a good knowledge of network protocol and various network technology. This paper aims to propose a method to automate this process.

1.1 Contribution

This paper describes a solution for automating the time-consuming and labor-intensive network analysis usually done manually by network administrators [24]. This automation will dramatically decrease the time required for analyzing PCAP files and allows less experienced administrators to check the files in the same way as a more experienced administrator would do.

To create a solution that can be easily used by network administrators, one needs to consider the following assumptions:

- Administrators often do not have enough programming skills;
- Adding support for new protocols should not require to create new protocol dissectors;
- Extending the tool with new detection capabilities must be straightforward for administrators;
- Administrators can work with packets as they are used to with manual analysis.

The presented solution can be compared to existing tools and other alternative approaches:

- Use an existing IDS or forensic tools: These tools are not made to work with network data which administrators usually work with. Adding new incident detection requires programming skills or understanding the nontrivial definition language. These tools usually have limited support of network protocols and protocol fields (even if the protocol is supported, not all fields are available). The usual workflow is to process each network flow individually to speed up the process which makes detection across multiple protocols

ECBS '19, September 2–3, 2019, Bucharest, Romania

ACM ISBN 978-1-4503-7636-5/19/09…$15.00
https://doi.org/10.1145/3352700.3352714

and flows more complicated. More details about existing solutions are in Section 3;
- Implement a new analysis tool from scratch: Creating a complete tool would require a lot of effort (time, people, money) and even after the tool would have been created, it would require additional human resources to add support for new protocols or new protocol versions;
- Use the machine learning approach [17]: This approach would require programming skills from administrators while extending the tool and the approach would be more like a magic black box instead of a straightforward approach.

It is really important to clarify that our goal is not to create another IDS tool or to create an alternative to existing IDS tools (these tools are more described in Section 3). IDS tools already contain a huge amount of predefined rules to detect well-known attacks and are highly optimized to maximize performance. Our goal is to create a complementary tool which would be used alongside existing IDS tools. This tool would allow administrators to automate their manual incident analysis by specifying incidents using an easy to understand and use threat specification language.

2 SECURITY ANALYSIS OF NETWORK COMMUNICATION

The goal of the network security analysis is to detect security incidents inside the network and provide as much information about these incidents as possible [11]. Example of such information can be the source, progress, and consequences of the incident. Finding evidence for an incident is also called a forensic analysis.

In our example, the Internet is not working because the victim's computer cannot be assigned an IPv6 address. Some other device is blocking the address assignment, and after several attempts, the victim's computer gives up. As illustrated in Figure 1, the reason for this is that an attacker applies a denial of service attack by misusing the *Duplicate address detection* mechanism.

```
Neighbor Solicitation for 2001:abcd::d45e:169a:fbe7:921f
Neighbor Advertisement 2001:abcd::d45e:169a:fbe7:921f (ovr) is at 00:0c:0a:d8:1d:ab
Neighbor Solicitation for 2001:abcd::6064:dec3:35e8:3bb0
Neighbor Advertisement 2001:abcd::6064:dec3:35e8:3bb0 (ovr) is at 00:0c:41:d3:68:16
Neighbor Solicitation for 2001:abcd::800f:ddfb:3666:8f1b
Neighbor Advertisement 2001:abcd::800f:ddfb:3666:8f1b (ovr) is at 00:0c:1f:3e:35:9c
Neighbor Solicitation for 2001:abcd::1193:1c8c:691f:977e
Neighbor Advertisement 2001:abcd::1193:1c8c:691f:977e (ovr) is at 00:0c:01:91:a6:a7
Neighbor Solicitation for 2001:abcd::2c21:184:d2db:d8a1
Neighbor Advertisement 2001:abcd::2c21:184:d2db:d8a1 (ovr) is at 00:0c:f1:6d:a4:ca
```

Figure 1: The output from the Wireshark tool. The client is trying to assign a unique address, but each time the client checks the availability of the address with the Neighbor Solicitation message, the attacker blocks its assignment by sending the Neighbor Advertisement message.

The primary purpose of the gathering information about the incidents is to reduce damage to the affected company owning the network infrastructure and services [10]. The harm can be caused by interrupting normal business processes, data breach, wasting resources exploited by a botnet. After the incident is solved, it is very important to prevent similar incidents from happening again.

Each analysis should consist of several actions. It is necessary to find out who or what is the source and destination of the attack (or generally incident) [11]. The source can be a corrupted device inside the network or person on the other side of the globe. Another action is identifying exploited vulnerabilities and finding the solution for fixing them. It is also necessary to identify which parts of the network were exposed by the attack and can potentially be abused.

Is target IPv6 address available to use?

src IPv6	- ::
dst IPv6	- ff02::1::ffe7:921f
ICMPv6 type	- Neighbor Solicitation
target address	- 2001:abcd::d45e:169a:fbe7:921f

No, you can't. I'm using that address.

src IPv6	- 2001:abcd::d45e:169a:fbe7:921f
dst IPv6	- ff02::1
ICMPv6 type	- Neighbor Advertisement
target address	- 2001:abcd::d45e:169a:fbe7:921f

Figure 2: In our example, we can see that for each Network Solicitation (NS) message an adequate Network Advertisement (NA) message is received. All NA messages have a different source IPv6 address. The administrator should find the location of this device and continue with the investigation by further analysis of the device's activities.

From a technological point of view, the security analysis is problematic because it requires information about transferred data inside the network. One possible approach is to capture all packets and perform full stack analysis. The problem is that networks usually transfer a lot of data and it is impossible to analyze all data in necessary detail. Another approach is to ignore layer 7 information completely and use only data from lower layers, for example, Netflow data [9]. With this approach, it is possible to process a huge amount of data traffic, but the analysis is limited only to data from lower layers.

The analysis is problematic from a personal point of view too. Computer networks consist of many network devices, protocols, applications, and the analysis process requires good knowledge for all of these elements. Even if an analyst has the necessary knowledge, she must understand the analyzed network very well, because each network environment is different and has some specific properties. Another problem is that the analysis can be a very time-consuming activity even for a skilled administrator. On the other hand, the analysis should be finished as soon as possible to prevent the incident from doing more damage [10]. This puts an administrator under time pressure which can lead to mistakes.

From our point of view, the network analysis process should be split into two parts:

- **Detect possible incident** - By using less complex analysis, it is possible to constantly analyze all the transferred traffic and search for an anomaly. E.g., anomaly detection over Netflow data, IDS system over transferred data payloads or keywords detection inside the server log files [26]. Usually, most of the anomalies will not be a security incident, but

just a false-positive detection. Based on the anomaly parameters, like IP addresses, a small portion of network traffic is captured and saved into a PCAP file for further analysis.
- **Execute complex analysis** - Captured PCAP file is analyzed in detail to determine if the anomaly is a network security incident or not. This process is usually done manually by an administrator, and it is very time- and labor-consuming. The result of this step is a report of a security incident with as much information as possible. Emphasis is on accurate results, not quick results.

3 RELATED WORK

Intrusion detection systems (IDS) employ different mechanisms to detect potentially harmful communication. Rule-based IDS systems use predefined patterns (signatures) that match suspicious packets in network communication. The signature is a structured set of rules that is used to identify an attack [14]. Attacks are detected by searching for these signatures in packets. Anomaly-based systems are based on a creation of long-term network traffic statistics, or the use of artificial intelligence to obtain a common network traffic model [6]. When the deviation of the actual communication from the created profile is detected a system reports an attack alert. IDS are optimized to perform a real-time detection; thus the complexity and expressive power of the detection rules are limited. While IDS produces an alert for a detected attack or abuse, the burden of further work falls on the analysts who must collect evidence within the network traffic data. This is usually a series of manual activities during which the analyst filters captured traffic, decodes the packets and collects relevant information from the packet's header and payload. Research has been done towards providing automated analysis procedures or aiding the process by visual analysis methods.

3.1 Intrusion Detection Systems

One of the most widespread IDS tools is *Snort* [1]. The system allows administrators to search by string, binary data or regular expression. The search is performed either in data within the same transport streams or on a per-packet basis. *Snort* does not deal with the structure and semantics of application protocols. With *Snort*, we are not able to find if there is an LDAP packet with the resulting code 90 that represents memory problems. If the system tries to find just a value of 90 (or hexadecimal 0x5A), the result would contain many false positives.

The*Suricata* tool [2] has support for application protocol decoding, although the list of supported protocols is quite small (around 13 protocols). Also, for the supported protocols, the tool defines only very few fields. For example, only four fields are defined for protocol ICMP, and for DNS protocol only one field is defined. On the other hand, *Wireshark* defines up to 79 ICMP fields and 298 DNS fields.

The *Bro* tool [21] (currently known as Zeek [3]) performs a deep packet inspection of the traffic to detect known security threats. Zeek performs an in-depth analysis of network communication keeping an application-layer state which enables it to perform a more advanced analysis in comparison to traditional signature-based IDSs. An event-driven scripting language makes it possible to customize the system to one's specific needs. The idea behind Bro/Zeek is similar to the proposed system, but it differs in several aspects. Firstly, extending the system with a new application protocol requires writing a new dissector, thus, similarly to *Suricata*, Bro/Zeek supports just around 50 protocols, and not all protocol fields are supported. Secondly, writing Bro scripts requires advanced programming skills often not had by average network administrators. Thirdly, while the scripting language is very powerful, to write rules for non-trivial cases is not straightforward.

3.2 Network Traffic Analysis

Network traffic analysis corresponds to the examination of network communication for the purpose of computer security, troubleshooting and system debugging. The most commonly used tool for manual network traffic analysis is *Wireshark* [4] [2, 18, 19]. *Wireshark* supports decoding of all standardized and widely used network protocols. The shortcoming of the *Wireshark* tool is that it lacks any advanced automation [5]. Also, the tool cannot provide a big picture view of the data without cognitive overloading of the user [8]. To overcome tedious work of network traffic analysis the analysts and operators rely on automation scripts that reduce cost and cut down the time required to complete the investigation. Information visualization aids security analysts in detecting anomalous events by increasing their situational awareness through the visual representation of network flow [13]. Another tool, VisAlert, provides an extensible visualization that can accept multiple data sources, including IDS alerts and system log files [15]. Krasser et al [12] proposed a tool that uses 3D animation to provide rich visualization information on network communication.

When the traffic is encrypted the analyst cannot apply the traditional approach based on decoding and examination of individual packets. In such a situation, traffic characterization is often the only viable option. Encrypted traffic is characterized into different categories, according to the type of traffic e.g., browsing, streaming, etc. To do this, statistical or machine learning methods are often used [4]. This approach is also effective for identification of malware from the network behavior [22].

3.3 Rule-based Methods

There are also other network domains where administrators use rule-based systems, for example in SCADA networks management [25] or troubleshooting [16]. Also, these systems share the same limitations: per-flow or per-packet analysis, they are not easily extendible, or they are using a language unknown to a typical administrator. The rule-based approach is, of course, practicable also outside the domain of computer network security. The primary purpose of the tool *Yara* [5] is the detection and classification of malware samples in various sources (e.g., files, folders, processes). Analysts can create rule sets based on the textual or binary patterns that are transformed into a compiled form and then used for the search in the specified entities. *Yara* can detect predefined patterns in the files, but it does not provide a way how to more

[1] http://www.snort.org
[2] https://suricata-ids.org/
[3] https://www.zeek.org/
[4] https://www.wireshark.org/
[5] https://virustotal.github.io/yara/

comprehensively describe the relations between the data, and it does not involve its structure and semantics. The Yara tool was used for file system analysis as well as for memory analysis [3].

The proposed system is similar to the signature-based IDS but has some significant differences. Signatures for IDS systems are typically generated by the system creator or the community around that system. To create a new signature, a significant effort is required [7]. Another limitation of IDS systems is that they work on the flow-level [20]. Systems process each flow independently, and therefore they usually search for events within an individual stream. However, it is very also necessary to be able to look for the data across streams, because there are threats which distribute the traffic across multiple streams, e.g., malware which uses a multi-band technique to hide from the detection [1]. Our proposed system, however, can easily detect events spread across multiple streams.

4 RULE-BASED NETWORK EVENTS FINDER

The proposed tool performs automatic analysis of packet traces of network communication. The tool is driven by the collection of rules that when evaluated are able to identify if a certain event happened. As described in Section 2, we focus primarily on network incidents, thus any identified event stands for the identified security incident. However, the tool can also be used in other areas, such as Network Diagnostics or Performance Monitoring. For this reason, instead of referring to incidents, we use the more general term *event* to describe the information we are looking for inside the captured network packets.

The overall architecture of the system is illustrated in Figure 3. With this architecture, an administrator will write event descriptions in human-readable configuration files which the tool will convert into an executable format (step 1). An administrator can easily add, modify and delete event descriptions. After the tool converts all configuration files, an external tool is used to convert captured packets into the format suitable for further processing and searching in them (step 2). Lastly, the tool takes converted event descriptions one by one and tries to search for them inside the converted packets (step 3). Based on the findings, the tool generates an appropriate report.

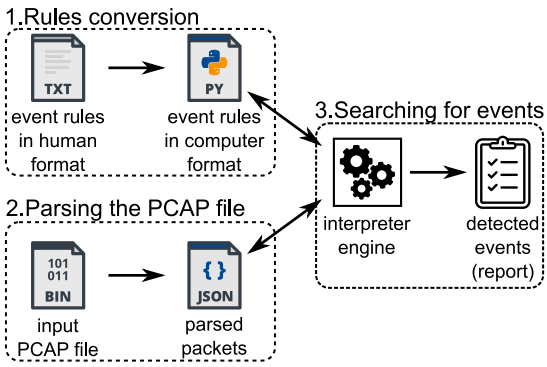

Figure 3: An architecture and workflow of the proposed tool.

The rest of this section (i) describes the format and language of the event description files, (ii) discusses parsing of the input PCAP files, and (iii) explains how the tool searches for events inside the input data.

4.1 Rules language

The rule language is used by users to specify the matching conditions and actions related to identified events. The rule language represents a compromise between easy to use representation and expressivity. Also, for expressions, the *Wireshark*'s display filter language is utilized [6]. It has two fold reasons. Firstly, the display filter language is expressive enough to represent simple queries to a collection of packets. Secondly, language is well-known and easy to understand among network administrators.

Each event rule consists of a name, description, and the body consisting of five attributes: *group*, *packets*, *asserts*, *threshold* and *report*. Figure 4 shows an example of a rule, which describes how to detect the SLAAC Duplicate Address Detection (DAD) attack. The details and explanation of the meaning for each attribute is given in the next subsection.

```
1  name: SLAAC DAD
2  description: Possible Duplicate Address
       Detection attack, find attacker from
       NA source MAC address.
3  group:
4    - icmpv6.nd.ns.target_address icmpv6.nd.
       na.target_address
5  packets:
6    –NS: icmpv6.type==135 and ipv6.src=="::"
7    –NA: icmpv6.type==136 and ipv6.dst=="::"
8  asserts:
9    - count(NS) > 0 and count(NA) > 0
10 threshold: 5
11 report: packets
```

Figure 4: Event description for detection of the SLAAC DAD attack.

4.1.1 Name and description. Name is an identification of the rule which needs to be unique. A description is used to describe the event in a human-understandable format. Both the name and the description are part of the created report.

4.1.2 Groups. The section *group* specify how the packets are split into several disjunctive groups. The tool assigns packets into groups based on the value of specified protocol fields. For example, when the administrator specifies the source IP address (ip.src) as a protocol field, each IP address will have a separated group containing packets which that IP address sent.

Figure 5 shows the format of the rule part *group*. If more than one protocol field is specified on a single line, the tool takes protocol fields one by one in the same order as specified and tries to find the values for these fields. This process stops when the value is found or when the packet does not contain any protocol value from the line. If more than one line is specified, a packet must have

[6] https://www.wireshark.org/docs/dfref/

at least one value for each line, and in that case, the tool marks the group as a list of values (one line = one list element). Otherwise, the packet will be ignored in future processing, because it does not belong to any group. One exception is when the rule does not contain any line and any protocol field. In that case, the tool assigns all packets into a single "default" group.

```
group:
 - field_1 field_2 ... field_N
 - ...
-----------------------------------------------

# field_X - any Wireshark's protocol field
    name
```

Figure 5: Format of the event's group section

4.1.3 Packets. Inside each group of packets, the tool tries to find predefined packets. These predefined packets have a special meaning for the event detection (e.g., special protocol field value or amount of these packets). Each packet has its name and filter definition. The result is a list of packets fulfilling the filter.

Figure 6 shows the format of the rule part *packets*. Each packet is defined by its unique name and by the *Wireshark*'s display filter language. This is the same filter which a network administrator would use during a manual analysis using the tool *Wireshark*. Therefore, these filters can be copied from *Wireshark* or vice versa.

```
packets:
 - name_1: filter_1
 - ...
-----------------------------------------------

# name_X - the name will be used as a label
     for packets in section 'asserts'.
    Possible values: [a-zA-Z0-9_-]+
# filter_X - Wireshark's display filter
```

Figure 6: Format of the event's packets section.

4.1.4 Asserts. After the tool finished the search for the specified packets, the tool evaluates the assert conditions. These conditions define if the expected state (e.g., too many packets) was detected inside the group of packets. The assert language is based on the *Wireshark*'s display filter language, but contains three improvements:

(1) possibility to get the number of detected packets. This is implemented with function *count*() which takes the name of the specified packet (from the Section 4.1.3) as the parameter. E.g., $count(DNS) > 100$;
(2) it is possible to use basic mathematical operations (+, -, *, /). E.g., $count(DNS)/count(all) > 0.5$;
(3) searching for a specific field name can be limited only on specified packets from the section packets. E.g., $DNS[udp.port] == 53$ will try to find value udp.port only inside detected packets with name DNS. If no packet name is specified, the search is focused on all packets in the group.

Figure 7 shows the format of the event's asserts part together with the three improvements. Name 'count' is the name of the function, *packets_name* is the name of the defined packets from the event packets section, and *field_name* is the name of the protocol field from the *Wireshark*'s display filter language.

```
asserts:
 - condition_1
 - ...
-----------------------------------------------

# condition_X - Wireshark's display filter
    with possible extensions:
# 1) count(packets_name) - used as a
    constant
# 2) math_sign - used as a operator
# 3) packets_name[field] - used as a
    variable
# count() - name of the function.
# packet_name - name from part 'packets'.
# math_sign - one of the following
    mathematical signs: +, -, *, /
# field - Wireshark's protocol field name
```

Figure 7: Format of the event's asserts section.

4.1.5 Threshold. Each group which fulfills all assert conditions increases a counter designated to count all fulfilling groups. This counter is compared with a threshold and based on whether it is equal or greater than the threshold, the tool generates a report.

Figure 8 shows a very simple format of the threshold. Only one integer value is specified.

```
threshold: value
-----------------------------------------------

# value - Numeric contant. Possible values:
    [0-9]+
```

Figure 8: Format of the event's threshold section.

4.1.6 Report. Section *report* specifies how detailed the generated report will be. For example, it is not useful to export all packets which are part of a large DDoS attack. Three levels are defined: *event* - only data from the section threshold are used, *groups* - all group values which contribute to the final report are added and *packets* - each group reported will also contain a list of all the packets inside that group.

Figure 9 shows the format of the report section. Only one string with one of the three predefined values is defined.

```
report: level
-----------------------------------------------

# level - Specifies the detail of generated
    report. Possible values: event, groups,
    packets
```

Figure 9: Format of the event's report section.

4.2 Packets parser

The external tool called *TShark* [7] (command line version of the *Wireshark* tool) is used to parse captured network communications. We use an existing tool to avoid the necessity of creating parsers (or dissectors) for individual protocols. As *TShark* provides the same set of dissectors as *Wireshark*, we can immediately support almost all current network protocols in the rules. Using *TShark* approach comes with several pros and cons:

+ using an external well-maintained tool decreases the requirements for the created tool;
+ tunnel data can also be processed as if the tunnels were not used at all;
+ supports a large number of protocols (over 3000) as well as the number of fields that can be obtained from the protocols (over 227000) - valid for version 2.6.3.
+ processed data can be exported into the JSON format so that later processing can directly access the values of the protocol fields;
+ the fields of the individual protocols are labeled in the same way as in *Wireshark*, so the administrator does not need any special documentation to find the names of the fields;
+ *TShark* does not just parse the data, it also analyses it. For example, it detects packet retransmission or calculates application statistics;
- if some protocol is not supported by *TShark* (usually a proprietary protocol), adding support for it requires very good programming skills;
- *TShark* is not fast at analyzing large PCAP files (gigabytes and more).

The description and options of *TShark* can be found in its documentation. Just for clarification of how the JSON from the *TShark* looks like, Figure 10 shows an example of the output. The JSON contains protocol field values from all network protocols in a key-value data format. Hierarchy of the JSON data attributes directly represents the structure of the packet protocols.

```
...
"eth": {
  "eth.dst": "f0:79:59:72:7c:30",
  "eth.src": "00:1f:33:d9:81:60",
  "eth.type": "0x00000800",
},
...
"udp": {
  "udp.dstport": "53",
  ...
},
...
"dns": {
  "dns.id": "0x00007956",
  "dns.flags.response": "0",
  "dns.qry.name": "mail.patriots.in",
  ...
},
...
```

Figure 10: Excerpt from the TShark example output.

[7] https://www.wireshark.org/docs/man-pages/tshark.html

4.3 Rule checker

When the tool converts PCAP file and all rules into a format suitable for further processing, the tool starts searching for the events with the last part of the architecture - interpreter engine. The interpreter engine takes the rules one by one and tries to find whether some rule is matched inside the data or not. Based on the rule definition, the engine creates an adequate report. The engine consists of five parts as shown in Figure 11. Each part corresponds to one section from the event rule definition (Section 4.1). This is illustrated using boxes in the Figure together with rule lines which are copied from the rule definition example in Figure 4. Following subsections describe these five engine parts.

4.3.1 Grouping packets. The goal of the packet grouping part is to separate packets into groups according to the specified attributes (see Section 4.1.2). If the packet has multiple values for the defined attribute, the same packet will be placed into multiple groups. This separation into groups allows processing of packets from one group independently from packets in other groups. The attribute can be any field name which *Wireshark* defines. Example of such an attribute is a source IP address ($ip.src$) or TCP stream index ($tcp.stream$). In the case the packet does not contain a specified attribute, the tool ignores it during further processing. There is one exception to this rule, the situation when an administrator does not specify any attribute at all. In that situation, all packets are part of the same "empty" group.

It is possible to specify more than one attribute for dividing packets into groups. There are two modes in which an administrator can specify multiple attributes - AND and OR and these modes can be combined. In the OR mode, the tool is trying to find the first attribute inside the data and continues looking for other attributes only if there is no match. To continue the processing of the packet, the tool must find at least one attribute. With the AND mode, the tool must find all specified attributes. Another difference is that the tool will identify these groups using a list of values instead of one value.

Figure 11 shows an example with two attributes in OR mode (both are specified in the single row as can be seen in Figure 4): $icmpv6.nd.ns.target\_address$ and $icmpv6.nd.na.target\_address$. These attributes group SLAAC *Neighbor Solicitation* (NS) messages with *Neighbor Advertisement* (NA) messages according to the requested address. Example of group identification is value $2001 : abcd :: 6064 : dec3 : 35e8 : 3bb0$.

4.3.2 Searching specific packets inside groups. After the tool assigns each packet into the appropriate group, the tool tries to search for specific packets inside them. All packets fulfilling the search condition are returned. As described in Section 4.1.3, packets are specified using the *Wireshark*'s display filter language, and their evaluation is the same as in *Wireshark*.

In the example from Figure 11, the tool tries to find two sets of packets (NS and NA) specified by conditions $NS = \ldots$ and $NA = \ldots$, For the group $2001 : abcd :: 6064 : dec3 : 35e8 : 3bb0$, the NS will contain all NS messages which check the availability of address $2001 : abcd :: 6064 : dec3 : 35e8 : 3bb0$ and NS will contain replies for NS messages.

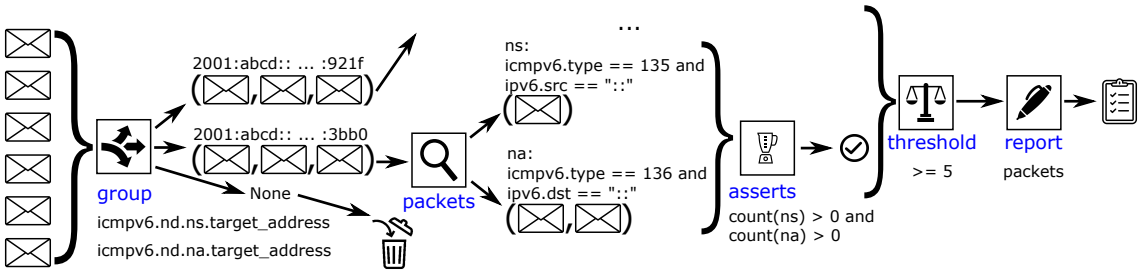

Figure 11: The rules evaluation engine which consists of five parts. On the left side are parsed packets which are the input of the engine and the output is report placed on the right side. Bellow each part are lines from the example code in Figure 4.

4.3.3 Checking asserts for each group. The event can contain several assert rules. All of the assert rules must be valid to consider a group as the group which fulfills the assert rules. Checking whether assert rules are valid or not is very similar to searching packets in groups. As was described in Section 4.1.4, asserts use an extended the *Wireshark*'s display filter language. This language extension modifies how the tool evaluates the assert filter. There are three language extensions:

(1) It is possible to work with information about how many packets the tool detected inside a group. This information is calculated using the function *count*(*packet_name*). Before the evaluation of the assert conditions, all these functions are executed and replaced by their results (numeric constant).
(2) When looking for a specific field inside the packets, *Wireshark* uses just the name of the protocol field (e.g., *ip.src* = 8.8.8.8). When this code is placed in the assert condition, the tool will check only packets from the group. However, it is also possible to limit these searches for values only to packets which the tool found in the packets section. For example, during the asserts evaluation with the code *query*[*ip.src*] = 8.8.8.8, the tool will supplement only packets fulfilling the query filter.
(3) An administrator can use basic mathematical operations with the *Wireshark* fields, functions and constants. For example, an administrator can use them for calculating the ratio between two sets of packets. With the code $count(X)/count(Y) > 2$, the tool evaluates the condition as true only if the amount of packets fulfilling the packet filter X is at least twice as many as packets fulfilling the packet filter Y.

In the example in Figure 11, the assert rule is $count(NS) > 0$ *and* $count(NA) > 0$, which specifies, that there must be at least one NS message and one NA message for the same queried IPv6 address.

4.3.4 Counting the group matches. After the tool evaluates each asserts condition for each group, the tool checks the amount of the fulfilling groups. The result is a single numeric value which the tool compares with a defined threshold. If the threshold condition is met, the tool located the event in the input data and starts the generating report process.

In the example from Figure 11, we are counting how many verified addresses ended as duplicated. If this number is equal to 5 or higher, the tool detects the DAD attack.

4.3.5 Generating report. If the tool detects an event, it generates an appropriate report. The report can have three levels of detail (as described in Section 4.1.6): *event* (minimum details), *group*, and *packets* (maximum details).

Figure 12 shows a report from our DAD attack example with three vertical lines which illustrate, which lines the report would contain with a different level of detail. This is also a result of the example introduced in Figure 1; the tool detected SLAAC DAD attack.

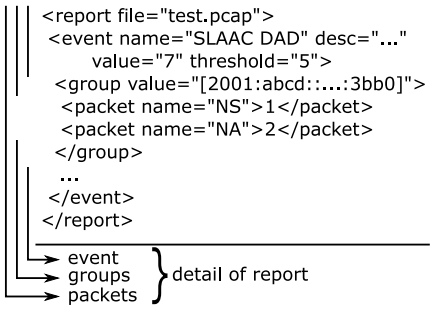

Figure 12: The output of the tool which contains a detected DAD attack. The vertical lines represent how the report would look with different requested detail.

5 CONCLUSION

Packet-level network analysis is a very time-consuming activity requiring good knowledge of network protocols. An administrator usually uses the packet analyzer which performs capturing network data, dissecting network packets and providing other related information. This paper describes an approach to automating this process.

Signature-based IDS tools provide similar functionality to our tool. The problems with IDS systems are that they usually require

programming knowledge for extending their capabilities, they support only a limited number of protocols and their detection capabilities are limited to flow-scope or packet-scope analysis.

We have proposed an interpreter architecture, which uses a rule-based approach for defining security events. We are using *TShark* to parse the input data, which removes the necessity of writing custom protocol parsers. The tool uses a format which was inspired by the *Wireshark*'s display filter language. This format allows administrators to create or modify a rule without any programming skills with the knowledge they already have. The tool is supposed to fill in a missing spot for automation framework for security analysts that must analyze packet traces related to the incident based on alerts generated by IDS tools. The presented tool automatically performs the analysis using defined rules and identifies the location using evidentiary material.

Most of the currently used tools search for data on a per-packet or per-flow basis. Our tool is capable of searching for events across multiple flows without any limitations which makes the searching process more flexible. Because searching without these limitations is very expensive, the tool is not designed to fully replace already existing IDS solutions which are less flexible, but capable of processing much more data.

We have created a proof-of-concept implementation, which covers the entire analysis process. The event rule files are transformed into a format suitable for further processing. After that, the tool uses the external tool *TShark* (command line version of the *Wireshark* tool) to parse the input PCAP file and to save them into the JSON format. The final stage of the tool takes this JSON file and applies available rules to identify the security threats in the input data. At the end of the analysis, the tool generates a report.

To demonstrate the functionality of the implemented prototype, we have created a specification of 35 different security events, e.g., MitM ARP attack, HSRP protocol configuration with nonoptimal configuration, network scanning, using old-unsecured TLS version and so on.

The future work will focus on enriching generated output. For example, the output should contain a *Wireshark*'s display filter expression which can be used by an administrator for manually confirming the detected event in *Wireshark*. Also, the language should be enriched with new features, for example, usual statistical and aggregation functions. To be practically usable, optimizing the performance of the tool is necessary.

ACKNOWLEDGMENTS

This work was partially supported by the BUT FIT grant FIT-S-17-3964, "ICT tools, methods and technologies for smart cities".

REFERENCES

[1] MITRE ATT&CK. 2019. Technique: Multiband Communication. https://attack.mitre.org/techniques/T1026/

[2] Laura Chappell. 2017. *Wireshark 101: Essential Skills for Network Analysis-Wireshark Solution Series.* Laura Chappell University, USA.

[3] Michael Cohen. 2017. Scanning memory with Yara. *Digital Investigation* (2017). https://doi.org/10.1016/j.diin.2017.02.005

[4] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. 2016. Characterization of Encrypted and VPN Traffic using Time-related Features. In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy - Volume 1: ICISSP.* 407–414. https://doi.org/10.5220/0005740704070414

[5] Alia Yahia El Sheikh. 2018. Evaluation of the capabilities of Wireshark as network intrusion system. *Journal of Global Research in Computer Science* 9, 8 (2018), 01–08.

[6] Pedro Garcia-Teodoro, Jesus Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security* 28, 1-2 (2009), 18–28.

[7] Ibrahim Ghafir, Vaclav Prenosil, Jakub Svoboda, and Mohammad Hammoudeh. 2016. A survey on network security monitoring systems. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW).* IEEE, 77–82.

[8] John R Goodall, Wayne G Lutters, Penny Rheingans, and Anita Komlodi. 2006. Focusing on Context in Network. *Security* April (2006), 72–80.

[9] Rick Hofstede, Pavel Čeleda, Brian Trammell, Idilio Drago, Ramin Sadre, Anna Sperotto, and Aiko Pras. 2014. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys & Tutorials* 16, 4 (2014), 2037–2064.

[10] Computer Economics Inc. 2007. 2007 malware report: The economic impact of viruses, spyware, adware, botnets, and other malicious code. http://www.computereconomics.com

[11] Karen Kent, Suzanne Chevalier, Tim Grance, and Hung Dang. 2006. Guide to integrating forensic techniques into incident response. *NIST Special Publication* 10, 14 (2006), 800–86.

[12] Sven Krasser, Gregory Conti, Julian Grizzard, Jeff Gribschaw, and Henry Owen. 2005. Real-time and forensic network data analysis using animated and coordinated visualization. In *Proceedings from the 6th Annual IEEE System, Man and Cybernetics Information Assurance Workshop, SMC 2005.* https://doi.org/10.1109/IAW.2005.1495932

[13] Kiran Lakkaraju, William Yurcik, and Adam J Lee. 2004. NVisionIP: NetFlow Visualizations of System State for Security Situational Awareness. *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security - VizSEC/DMSEC '04* (2004). https://doi.org/10.1145/1029208.1029219

[14] Hao Li, Guangjie Liu, Weiwei Jiang, and Yuewei Dai. 2015. Designing snort rules to detect abnormal dnp3 network data. In *2015 International Conference on Control, Automation and Information Sciences (ICCAIS).* IEEE, 343–348.

[15] Yarden Livnat, Jim Agutter, Shaun Moon, Robert F. Erbacher, and Stefano Foresti. 2005. A visualization paradigm for network intrusion detection. In *Proceedings from the 6th Annual IEEE System, Man and Cybernetics Information Assurance Workshop, SMC 2005.* https://doi.org/10.1109/IAW.2005.1495939

[16] GeokHong Phua Lihui Chen Ming Luo, Danhong Zhang. 2011. An interactive rule based event management system for effective equipment troubleshooting. *Proceedings of the IEEE Conference on Decision and Control* 8, 3 (2011), 2329–2334. https://doi.org/10.1007/s10489-005-4605-0

[17] Srinivas Mukkamala and Andrew H Sung. 2003. Identifying significant features for network forensic analysis using artificial intelligent techniques. *International Journal of digital evidence* 1, 4 (2003), 1–17.

[18] Vivens Ndatinya, Zhifeng Xiao, Vasudeva Rao Manepalli, Ke Meng, and Yang Xiao. 2015. Network forensics analysis using Wireshark. *International Journal of Security and Networks* 10, 2 (2015), 91–106.

[19] Yoram Orzach. 2013. *Network Analysis Using Wireshark Cookbook.* Packt Publishing Ltd.

[20] Samuel Patton, William Yurcik, and David Doss. 2001. An AchillesâĂŹ heel in signature-based IDS: Squealing false positives in SNORT. In *Proceedings of RAID*, Vol. 2001. Citeseer.

[21] Vern Paxson. 1999. Bro: a system for detecting network intruders in real-time. *Computer networks* 31, 23-24 (1999), 2435–2463.

[22] Christian Rossow, Cj Dietrich, Herbert Bos, Lorenzo Cavallaro, Maarten Van Steen, Felix C. Freiling, and Norbert Pohlmann. 2011. Sandnet: Network Traffic Analysis of Malicious Software. *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS '11)* (2011), 78–88. https://doi.org/10.1145/1978672.1978682

[23] Sankardas Roy, Charles Ellis, Sajjan Shiva, Dipankar Dasgupta, Vivek Shandilya, and Qishi Wu. 2010. A survey of game theory as applied to network security. In *2010 43rd Hawaii International Conference on System Sciences.* IEEE, 1–10.

[24] Anna Cinzia Squicciarini, Giuseppe Petracca, William G Horne, and Aurnob Nath. 2014. Situational awareness through reasoning on network incidents. In *Proceedings of the 4th ACM conference on Data and application security and privacy.* ACM, 111–122.

[25] Yi Yang, Keiran McLaughlin, Tim Littler, Sakir Sezer, and HF Wang. 2013. Rule-based intrusion detection system for SCADA networks. (2013).

[26] Wang Zhenqi and Wang Xinyu. 2008. Netflow based intrusion detection system. In *2008 International conference on multimedia and information technology.* IEEE, 825–828.

A.4 Pattern Detection Based Network Diagnostics

Authors: Holkovič Martin, Ing. (50%), Bohuš Michal, Ing. (40%), Ryšavý Ondřej, doc. Ing., Ph.D. (10%)
Abstract: One of the most important parts of the network administrators' work is detecting and correcting errors inside computer networks. This part is also called network diagnostics. The problem is that computer networks are very complex, and there is no single universal approach for diagnosing the errors. In this paper, we propose a new method of diagnostics which utilizes looking for specific patterns inside captured network data. This approach automatically checks for all predefined patterns and generates a report with error descriptions for any detected errors. We have created a proof-of-concept tool and demonstrated its functionality.
Keywords: Network diagnostics, passive diagnostics, rule-based diagnostics, patterns lookup, patterns diagnostics, anomaly-based diagnostics.
Published in: 11th International Conference on Data Communication Networking (DCNET 2020), Setubal, Portugal
Conference rating: C(Core), B4 (Qualis)
ISBN: 978-989-758-445-9

Pattern Detection Based Network Diagnostics

Martin Holkovič, Michal Bohuš and Ondřej Ryšavý
Faculty of Information Technology, Brno University of Technology, Bozetechova 1/2, 612 66 Brno, Czech Republic
iholkovic@fit.vutbr.cz, xbohus01@stud.fit.vutbr.cz and rysavy@fit.vutbr.cz

Keywords: Network diagnostics, passive diagnostics, rule-based diagnostics, patterns lookup, patterns diagnostics, anomaly-based diagnostics.

Abstract: One of the most important parts of the network administrators' work is detecting and correcting errors inside computer networks. This part is also called network diagnostics. The problem is that computer networks are very complex, and there is no single universal approach for diagnosing the errors. In this paper, we propose a new method of diagnostics which utilizes looking for specific patterns inside captured network data. This approach automatically checks for all predefined patterns and generates a report with error descriptions for any detected errors. We have created a proof-of-concept tool and demonstrated its functionality.

1 INTRODUCTION

Computer networks are complex difficult to manage systems because they contain a large number of devices of different kinds that use a large number of different services and protocols. To fix any error that has occurred, it must be first identified and then correctly analyzed (Roy et al., 2010). Unfortunately, finding an error is not an easy task. Network administrators regularly spend a significant amount of time network troubleshooting (Zeng et al., 2012). Depending on the availability of the data, a diagnostic procedure can be done for the network traffic (Qadeer et al., 2010), application logs (Qiu et al., 2010), Netflow records (Garcia-Teodoro et al., 2009), etc. In this paper, we only consider network traffic, although the proposed method can work with other types of data sources.

An example of typical network issues is a problem with the DNS server resulting from a configuration change. Because of backup servers, the error may not immediately affect clients. One of the ways to detect the problem is to monitor the reply statuses of DNS queries from internal servers and determine the amount of successful and error replies. If the ratio between error and successful DNS replies increases significantly, an administrator is notified about the possible problem with the DNS server, e.g., the wrong configuration was inserted.

Detection of a peak within network data is currently implemented in most monitoring systems providing simple but efficient identification of anomalies. In this paper, we propose a system that can identify more complicated patterns in network data and attribute them to different errors. The data consists of a collection of timestamped events. For example, the amount of transferred data in the last 5 minutes. The proposed system analyzes the data and looks for predefined patterns that represent the specific situation, e.g., drop of transfer rate. The pattern search system uses simple descriptions of value changes, which are easily understandable by network administrators. An example of a pattern is a rapid drop followed by a sharp increase, which can be seen as a V-shape in the traffic graph. Administrators mostly use this form of visual analysis to get an overview of network status or to observe specific host behavior.

This paper's contribution is an automatic diagnostic of network problems by looking for patterns within timestamped events drawn from network traffic data. The method implemented in a proof-of-concept tool is demonstrated on data extracted from packet captures. However, the approach can also be applied to other suitable data sources, e.g., NetFlow data, logs, event files.

The structure of the paper is organized as follows. Section 2 discusses related work and describes similar approaches. Section 3 describes how the tool will be used. Section 4 describes the architecture of the proposed tool. Section 5 provides a simple tool usage demonstration. Finally, Section 7 contains the conclusion which summarizes the current state and proposes future work.

2 RELATED WORK

There are many ways to diagnose network errors. For maximum flexibility, it is necessary to be able to diagnose errors across TCP/IP layers (łgorzata Steinder and Sethi, 2004). In literature, this process is often referenced by authors as a root-cause analysis (RCA) (Solé et al., 2017). Although there is no standardized classification of diagnostic approaches, the most basic and well-known classes are deterministic and probabilistic (Solé et al., 2017).

One of the most common ways to diagnose network problems is by using the Wireshark tool (Ndatinya et al., 2015). However, Wireshark lacks any built-in automation, and therefore, efforts have been made to automate the work with this tool. One such tool implements decision trees that work with rules and exact matches similarly to the IDS systems (Holkovič and Ryšavý, 2019).

Our diagnostic approach, which consists of searching for patterns in data, is more similar to anomaly detection techniques. Anomaly detection is a search for situations when data are outside of the usual or expected value range (Chandola et al., 2009). As Martinez states in his work (Martinez et al., 2015), anomaly detection techniques can also be used for diagnostics purposes. An example is a solution that uses a neural network that learns anomalies and associated errors (Katasev and Kataseva, 2016). A similar approach was presented by (Ben Kraiem et al., 2019), they are also looking for patterns inside time-series. However, the patterns are limited to a single data series and a relationship between only one point and its directly adjacent points.

Similarly to diagnostics, the detection of anomalies is divided into classes, and the names and types of these differ in different literature sources. Compared to the diagnostics description, the closest classification is the anomaly (probabilistic) and signature (deterministic) class (Bhuyan et al., 2013; Kruegel and Toth, 2003; Sekar et al., 2002). Another possible classification is statistical, knowledge-based, and machine learning (Garcia-Teodoro et al., 2009).

Several approaches to analyze the network data, which are also usable for network diagnostics exist, for instance:

- *Intrusion detection system (IDS)* (Lee et al., 2005) Snort (Roesch et al., 1999; Li et al., 2015) or Bro (Udd et al., 2016) - they are looking for an exact match in transferred network data. These tools miss some diagnostic information.
- *Prudence* (Prayote and Compton, 2006) - a system which automatically learns the range of several network attributes and checks whether the current amount exceeds the ones in the learned model. This solution has a drawback that it is not able to detect anomalies for data traffic inside the learned ranges.
- *Entropy* (Gu et al., 2005) - the detection method uses maximum entropy technique to compare actual traffic with created baselines.
- *Signal processing* (Barford et al., 2002) - the data are split into several signals which are processed by special algorithms. The output from these techniques is hardly understandable by a regular network administrator.
- *Outliers* (Hodge and Austin, 2004) - These techniques look for variations in data. Part of the techniques deals with the detection of variations in graphs (Akoglu et al., 2015). These techniques use artificial intelligence (Rudrusamy et al., 2003) that makes the explanation of the diagnosed faults more complicated.

3 TOOL USAGE DESCRIPTION

Before describing the tool's architecture, we will describe how the tool will be deployed, how administrators will use it, and what kind of results the network administrator can expect from it. Further, this section describes which types of errors the tool can detect and the associated types of patterns.

The tool will not analyze traffic online but will work with captured PCAP files. When diagnosing a network problem, the administrator needs to capture the selected (problematic) network traffic and provides it as a PCAP file to the created tool. This implies that the tool will not perform all-time monitoring and that it will not be necessary to analyze full traffic across the network.

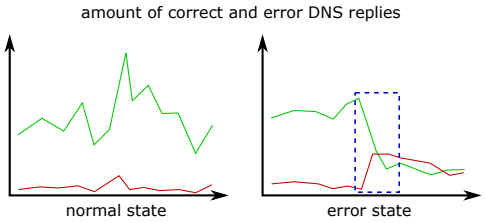

Figure 1: The figure illustrates how the problem with the DNS server configuration can be detected based on the network traffic visualization.

The tool's goal is to search for predefined patterns in selected packets in the specified PCAP file based on configuration files. Each such pattern is associated with a specific error in which the pattern occurs. An example is an error with the DNS server settings,

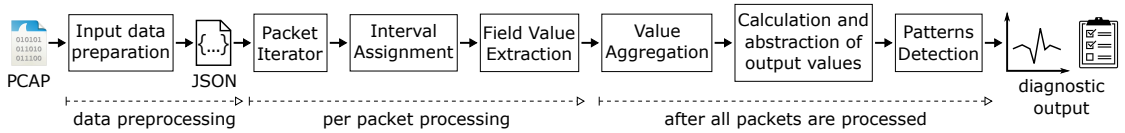

Figure 2: The architecture of the proposed tool, which consists of seven stages separated into three phases. The first phase prepares the input data, the second phase processes packets one by one, and the last phase begins when there are no more packets to process.

as mentioned in the introduction. A configuration error changes the ratio of correct and incorrect DNS responses. This example is shown in Figure 1.

The tool output will consist of detected patterns together with descriptions of detected errors to which the patterns relate. The detected pattern will also be drawn as an image file by the *Linux GNUplot* tool. Using these images, the administrator can easily identify whether it is a correct detection or a false positive.

The system will be able to detect two types of patterns - single and double. Simple patterns are searched in a single data series to detect fluctuations of values. For example, it is possible to detect incorrectly set QoS for a multimedia application or a significant increase in the average RTT value for the monitored server. Patterns will check whether the value decreases, increases, or stays the same.

The second type of pattern is double, which is searched for in two data series, analyzing the relationship between values from those series. It checks whether the value from one series is higher, less, or equal to the second series's value. With these patterns, it is possible to detect errors such as an increased rate of application errors or a non-functional load balance between two links to the Internet.

4 SYSTEM ARCHITECTURE

We have designed the tool as a single-thread application that consists of several parts. It is possible to implement the tool more efficiently as a multi-thread application executing several parts in parallel, but our tool is just a proof-of-concept of the proposed approach. The proposed system is displayed in Figure 2, and consists of seven stages, an input PCAP file and a configuration file.

The configuration file instructs the system which patterns should be detected. If more than one configuration file is specified, each configuration file is processed individually. There is only one exception, during the processing of the first configuration file, transferred data from the first stage is saved into the memory, so in the next configurations, this stage does not need to be executed. Configuration files use the YAML format, which is easily understandable by real users and also easily processed by computer programs.

4.1 Input Data Preparation

The first part of the tool prepares the input data for further processing. It begins with loading and parsing the PCAP file. This allowed us to use an already existing external tool called *TShark*. *TShark* is a command-line version of a well-known network management tool Wireshark which takes a PCAP file and converts it into the JSON format. The already implemented tool eliminates the need for implementing custom protocol parsers. *TShark* already supports hundreds of protocols, even if they are tunneled or segmented.

Another benefit of using *TShark* is that JSON output from the tool is marked by the Wireshark display language [1]. We have decided to use the Wireshark display language inside configuration files (specifically the inputs section). This well-known language will allow a better understanding of the configuration files by network administrators.

4.2 Interval Assignment

The X-axis of each chart represents the relative time when the packets were captured inside the input PCAP file. The time is represented by time intervals, where multiple values within one interval are processed in later stages, aggregated and represented by a single value.

The size of a time interval is specified in the config files in the section *interval* and has the format `"interval: value in milliseconds"`. It is up to the user to specify an interval adequately to the amount of data inside the input file so that the output chart will contain enough data to visualize a chart, and it will be possible to detect patterns in the chart.

During the processing of individual packets, the system calculates into which time interval the packets belong based on the field name *frame.time_relative*. The *frame.time_relative* value represents the number of seconds (with a microseconds accuracy) since the first packet inside the PCAP file was captured.

[1] https://www.wireshark.org/docs/dfref/

4.3 Field Value Extraction

The generated chart can be constructed from multiple values located in the source file, called the input series. For example, we may want to display the number of transferred bytes and the number of transferred packets. All input series need to be specified in the *input* configuration section.

The format of an *input* value is "`input_name: aggregation_function (field_name filter_condition)`", where:

- *input_name* - a user-defined name of the input series;
- *aggregation_function* - name of the aggregation function. The functionality is described in the following subsection. The possible functions are: MIN, MAX, AVG, COUNT, SUM, UNIQUE;
- *field_name* - field name from the Wireshark display language;
- *filter_condition* - an optional argument which consists of a comparator ($==$, $!=$, $<$, $<=$, $>$, $>=$) and a constant value (number or string).

The value is added to the list of values assigned to the calculated interval, only if the packet contains the specified field name and fulfills the filter condition. For example, if we would like to count only DNS packets with the domain name *server.local*, we would use `COUNT(dns.qry.name == "server.local")`.

4.4 Value Aggregation

After all packets are assigned into time intervals and their values are extracted, the system executes the aggregation function over all intervals and their values. The goal of the aggregation function is to replace a list of values with a single numeric value. For example, when an aggregation function *SUM()* is specified, the system iterates over the intervals and for each interval calculates a summary of the specified field name values. In case the interval does not contain any packets, the aggregated value is 0.

The process of interval assignment, field value extraction, and value aggregation is displayed in Figure 3. The packets from the input PCAP file are in the same order as they were saved in the PCAP file. The result is a list of aggregated values, however in case of the config file exports and aggregates multiple values at once, the result will consist of multiple lists.

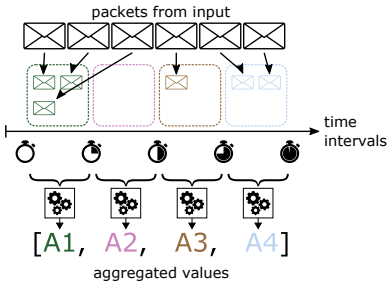

Figure 3: The figure shows how aggregation works. Packets from the input are grouped based on the time intervals, and an aggregation function is executed over each interval. The aggregation function returns a single numeric value.

4.5 Calculation and Abstraction of Output Values

A chart generated by the tool can be generated directly from the input series's values, or it is possible to calculate a series of new (output) values from multiple input series. For example, if we have one series containing HTTP bytes and another one containing HTTPS bytes, it can make sense to draw a chart as a summary of the two series. The *output* section of the configuration file specifies how the data is generated.

The section *output* contains a list of unnamed elements that can contain any name of the input series, mathematical operation $+, -, *, /$, and parentheses $(,)$. The output values are calculated based on input series values and a specified formula for each time interval separately.

The system is not searching for patterns in numerical values. The numerical values are abstracted by an alphabet character according to the type of patterns we are looking for. Two types of patterns exist:

- **simple** - patterns are evaluated based on a single output series only. There are five possible characters which describe original numerical values. The selected percentage levels were selected based on testing, and their purpose is to ensure that the detection does not depend on exact values, but allow minor variations.
  - *c - constant* - this value is used when the next value is the same as the previous value or within the range of $< 80\%; 120\% >$ of the previous value;
  - *r - raise* - the new value is larger, in the range of $< 120\%; 140\% >$ of the previous value;
  - *f - fall* - the new value is smaller, in the range of $< 60\%; 80\% >$ of the previous value;
  - *R - rapid raise* - the new value is significantly larger, in the range of $< 140\%; \infty\% >$;
  - *F - rapid fall* - the new value is significantly smaller, in the range of $< 0\%; 60\% >$;

- **double** - patterns are evaluated based on two different output series. There are three possible characters for them: *a (above), u (under), s (same)* describing that the first numerical value is larger (graphically above), smaller (graphically below) or approximately same (there is some tolerance) with comparison to the second numerical value.

At the end of the abstraction process, each output series will be represented by a single string with specific alphabet characters. This process of output value calculation and abstraction is displayed in Figure 4.

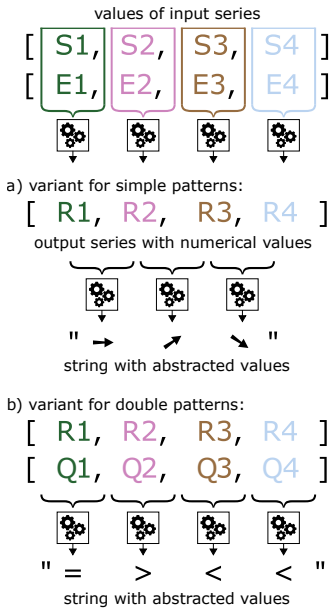

Figure 4: The figure shows the idea of calculating the numerical values of the output series and abstracting them into a string format. Two possible patterns are displayed - simple and double. In this figure, the string alphabet is replaced by arrows and comparator symbols for easy understanding.

4.6 Pattern Detection

The pattern lookup process is implemented as an evaluation of regular expressions over output series, which have a regular string format. The patterns are specified in the section patterns, and their format depends on whether we are working with a simple chart or a double chart:

- **simple** - the format is "`patterns: pattern1, pattern2, ...`". The table 1 lists all of the possible pattern names together with their regular expression definitions.
- **double** - the format is "`patterns: output_series_1 RELATION X% of output_series_2`". The X specifies the rate between the two compared values. For example, the `dns_error above 10% dns_success` rule is looking for a situation when the amount of DNS error packets will be at least 10% higher than the success DNS packets. The relation can have one of the following values:
  - *above* - detects the situation when the output series 1 is above the series 2;
  - *cross* - detects the situation when the output series 1 crosses the series 2 in whatever direction.
  - *under* - detects the situation when the output series 1 is below the series 2;

Table 1: Possible simple patterns and their definition in a regular expression format.

Pattern name	Regular Expression
rapidly raising	[R]+
rapidly falling	[F]+
tooth	[R][c]+[F]
reversed tooth	[F][c]+[R]
drop	[F][R]+[^F]
drop jump	[F][R]+[F]
peak	[R][F]

The idea of searching for patterns inside the abstracted data is displayed in Figure 5. The system is looking for a tooth pattern, which begins by a rapid raise, followed by at least one constant value and ends with rapid falling.

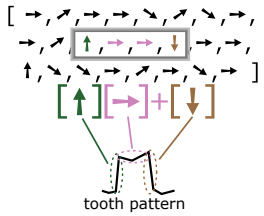

Figure 5: The idea of searching for patterns in the abstracted values.

5 EVALUATION

This section provides a simple demonstration of how to use the proposed tool. The tool always works with one PCAP file, which is prefiltered to contain only packets essential for analysis. So if the goal is to find patterns in network communication of several different applications, it is up to the network administrator to create one PCAP file for each application.

The example shows the transfer speed analysis of the selected application and attempts to find a *drop jump* pattern. This pattern looks very similar

to the electrocardiogram (EKG) signal. The drop jump pattern forms when the network is congested, and the router stores all unsent data into a buffer. The buffering will temporarily reduce the bit rate. After the network is no longer congested, the router sends the buffered data to their destination (Chappell and Aragon, 2014). Therefore, transmission speed temporarily raises above the normal level.

When diagnosing a problem with the poor quality of a video conference application, the *drop jump* pattern is searched for. When the pattern is detected, it means that the QoS queuing and buffering settings are applied to the application. These settings are undesirable, because they will decrease QoE (quality-of-experience) of such an application.

```
interval: 20
inputs:
 - transfer_speed: COUNT(frame.len)
outputs:
 - transfer_speed
patterns: drop jump
error: QoS queuing detected which
  may decrease the quality of user
  experience for multimedia traffic
```

Listing 1: The configuration specifies that the administrator is interested in the drop jump pattern applied at the transfer rate.

Listing 1 contains a configuration file that divides the input data into intervals with a length of 20 milliseconds and stores the amount of transferred data into those intervals. The transferred data are used as the input for the system, which tries to find the drop jump pattern. With the drop jump pattern, the system searches for a rapid drop, followed by a rapid increase in the value. Figure 6 shows an example of pattern detection in the PCAP file *"tr-queuing.pcapng"* from the *"Troubleshooting with Wireshark"* (Chappell and Aragon, 2014) book. The *error* describes what has been detected to the network administrator.

Before deploying the created tool inside the production network, it is necessary to evaluate the created tool using real data and to measure the accuracy of detection (true positive vs. false positive rate). In the case of a high false-positive detection rate, the level of the deviation will need to be adjusted to determine whether there has been an increase or decrease in the data values. Another option would be to create additional symbols to describe changes in values in more detail (e.g., increase by 10%, increase by 20%). However, more symbols would increase the complexity of the rules, and it would be harder for administrators to manage them.

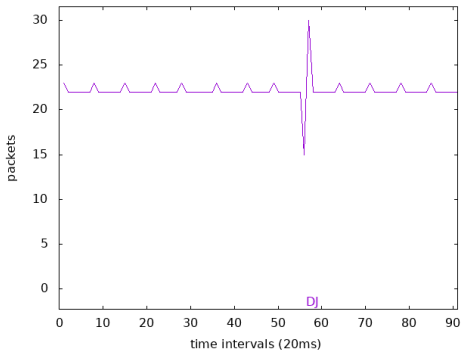

Figure 6: The output from the implemented tool which contains the detected drop jump (DJ) pattern as specified by the configuration in the Listing 1.

6 DISCUSSION

The presented method complements existing diagnostic tools for network troubleshooting. Because of its deterministic decision procedure based on the manually defined knowledge base, several advantages are provided:

- Repeated execution of the method for the same data yields the same results. This property is important for practical analysis when different paths during problem investigations are examined.
- Using the rule-based method, it is usually easy to observe the supporting information for the results presented. Rule execution can be traced to provide a path of reasoning followed by the system, which aids in understanding the issues and suggests possible corrective actions.
- The system is robust and flexible. It is possible to extend the system with new rules defined by a simple, declarative rule language.
- The system does not require a huge labeled data set to learn the classifier. It provides highly accurate detection for a carefully crafted collection of rules.

On the other hand, the method is limited in the following areas:

- It is not possible to identify behavior that is not represented in the knowledge base. If an attack exhibits a behavior, which has not been seen yet, it is not possible to detect it.
- Creating new rules can be difficult as sometimes it is hard to describe the expected situations in terms of packet count, size, and timing. While this is a very simple paradigm, it can represent a non-trivial class of network configuration issues and anomalies.

- To define the erroneous conditions, a deep knowledge of communication protocols and systems is necessary. Therefore the rules are to be defined by the domain expert. However, it may be possible to extend the system with specific rules identified by the network administrator using the rule language.
- The process of creating rules is mostly manual, and every update requires additional effort. However, to simplify the rule definition, an easy to understand declarative rule language was defined.

While modern methods introduced in the realm of computer network management stems from machine-learning algorithms, the rule-based approach is still prevalent in practice. It is because rules are easy to understand and rule evaluation is a deterministic procedure often offering enough information for finding the root cause of the issue by the administrator.

7 CONCLUSION

Network diagnostics is a complex activity requiring a lot of time and experience. We have presented a new rule-based approach to the detection and identification of network issues. The rules employ patterns that consist of a sequence of value changes to identify a sequence in network communication that possibly represents an anomaly. This new approach automates the labor activity conducted by network administrators that use the visual representation of network activities to identify non-standard situations.

We have implemented the proposed approach as a proof-of-concept tool that processes capture traffic and produces a log of identified issues. To demonstrate the functionality of the tool, we have tested the tool over a small amount of network data. The results confirm that the approach has practical potential, but further evaluation is required.

Future work will focus on: (i) Use this approach for another type of source data, such as log files or NetFlow records. It also makes sense to think about new types of patterns for these new data sources. (ii) Comparing the solution (accuracy and performance) with similar diagnostic tools. This could be difficult because each approach aims at different network errors, and accuracy will depend on created patterns and configurations. Also, many published papers on network diagnostics either do not provide access to the tools or datasets used for revaluation. (iii) Reimplementing the tool into pipeline architecture to allow the processing of real-time data.

ACKNOWLEDGEMENTS

This work was supported by the BUT FIT grant FIT-S-20-6293, "Application of AI methods to cyber security and control systems".

REFERENCES

Akoglu, L., Tong, H., and Koutra, D. (2015). Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29(3):626–688.

Barford, P., Kline, J., Plonka, D., and Ron, A. (2002). A signal analysis of network traffic anomalies. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 71–82.

Ben Kraiem, I., Ghozzi, F., Péninou, A., and Teste, O. (2019). Pattern-based method for anomaly detection in sensor networks. *21st International Conference on Enterprise Information Systems (ICEIS 2019)*, pages 104–113.

Bhuyan, M. H., Bhattacharyya, D. K., and Kalita, J. K. (2013). Network anomaly detection: methods, systems and tools. *IEEE communications surveys & tutorials*, 16(1):303–336.

Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58.

Chappell, L. and Aragon, J. (2014). *Troubleshooting with Wireshark: Locate the source of performance problems*. Laura Chappell University. ISBN: 978-1893939974.

Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., and Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1-2):18–28.

Gu, Y., McCallum, A., and Towsley, D. (2005). Detecting anomalies in network traffic using maximum entropy estimation. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 32–32.

Hodge, V. and Austin, J. (2004). A survey of outlier detection methodologies. *Artificial intelligence review*, 22(2):85–126.

Holkovič, M. and Ryšavý, O. (2019). Network diagnostics using passive network monitoring and packet analysis. *The Fifteenth International Conference on Networking and Services (ICNS)*, pages 47–51.

Katasev, A. S. and Kataseva, D. V. (2016). Neural network diagnosis of anomalous network activity in telecommunication systems. In *2016 Dynamics of Systems, Mechanisms and Machines (Dynamics)*, pages 1–4. IEEE.

Kruegel, C. and Toth, T. (2003). Using decision trees to improve signature-based intrusion detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 173–191. Springer.

Lee, H., Song, J., and Park, D. (2005). Intrusion detection system based on multi-class svm. In *International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing*, pages 511–519. Springer.

łgorzata Steinder, M. and Sethi, A. S. (2004). A survey of fault localization techniques in computer networks. *Science of computer programming*, 53(2):165–194.

Li, H., Liu, G., Jiang, W., and Dai, Y. (2015). Designing snort rules to detect abnormal dnp3 network data. In *2015 International Conference on Control, Automation and Information Sciences (ICCAIS)*, pages 343–348. IEEE.

Martinez, E., Fallon, E., Fallon, S., and Wang, M. (2015). Cadmant: Context anomaly detection for maintenance and network troubleshooting. In *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 1017–1022. IEEE.

Ndatinya, V., Xiao, Z., Manepalli, V. R., Meng, K., and Xiao, Y. (2015). Network forensics analysis using wireshark. *International Journal of Security and Networks*, 10(2):91–106.

Prayote, A. and Compton, P. (2006). Detecting anomalies and intruders. In *Australasian Joint Conference on Artificial Intelligence*, pages 1084–1088. Springer.

Qadeer, M. A., Iqbal, A., Zahid, M., and Siddiqui, M. R. (2010). Network traffic analysis and intrusion detection using packet sniffer. In *2010 Second International Conference on Communication Software and Networks*, pages 313–317. IEEE.

Qiu, T., Ge, Z., Pei, D., Wang, J., and Xu, J. (2010). What happened in my network: mining network events from router syslogs. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 472–484.

Roesch, M. et al. (1999). Snort: Lightweight intrusion detection for networks. In *Lisa*, pages 229–238.

Roy, S., Ellis, C., Shiva, S., Dasgupta, D., Shandilya, V., and Wu, Q. (2010). A survey of game theory as applied to network security. In *2010 43rd Hawaii International Conference on System Sciences*, pages 1–10. IEEE.

Rudrusamy, G., Ahmad, A., Budiarto, R., Samsudin, A., and Ramadass, S. (2003). Fuzzy based diagnostics system for identifying network traffic flow anomalies. *Proceedings of the International Conference of Robotics, Vision, Information and Signal Processing ROVISP*, pages 190–195.

Sekar, R., Gupta, A., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H., and Zhou, S. (2002). Specification-based anomaly detection: a new approach for detecting network intrusions. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 265–274.

Solé, M., Muntés-Mulero, V., Rana, A. I., and Estrada, G. (2017). Survey on models and techniques for root-cause analysis. *arXiv preprint arXiv:1701.08546*.

Udd, R., Asplund, M., Nadjm-Tehrani, S., Kazemtabrizi, M., and Ekstedt, M. (2016). Exploiting bro for intrusion detection in a scada system. In *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*, pages 44–51.

Zeng, H., Kazemian, P., Varghese, G., and McKeown, N. (2012). A survey on network troubleshooting. *Technical Report Stanford/TR12-HPNG-061012, Stanford University, Tech. Rep.*

A.5 Using Network Traces to Generate Models for Automatic Network Application Protocols Diagnostics

Authors: Holkovič Martin, Ing. (70%), Ryšavý Ondřej, doc. Ing., Ph.D. (20%), Polčák Libor, Ing., Ph.D. (10%)

Abstract: Network diagnostics is a time-consuming activity that requires an administrator with good knowledge of network principles and technologies. Even if some network errors have been resolved in the past, the administrator must spend considerable time removing these errors when they reoccur. This article presents an automated tool to learn the expected behavior of network protocols and possible variations. The created model can be used to automate the diagnostic process. The model presents a finite automaton containing protocol behavior for different situations. Diagnostics of unknown communication is performed by checking the created model and searching for error states and their descriptions. We have also created a proof-of-concept tool that demonstrates the practical potential of this approach.

Keywords: Network diagnostics, automatic diagnostics, protocol model from traces.
Published in: 10th International Conference on Data Communication Networking (DCNET 2019), Prague, Czech Republic
Conference rating: C (Core), B4 (Qualis)
ISBN: 978-989-758-378-0

Using Rule-Based Decision Trees for Automatic Passive Diagnostics of the Network Problems

Martin Holkovič

Flowmon Networks
Sochorova 3232/34
Brno 61600, CZ
Email: martin.holkovic@flowmon.com

Ondřej Ryšavý

Faculty of Information Technology
Brno University of Technology
Brno 61266, CZ
Email: rysavy@fit.vutbr.cz

***Abstract*—Network troubleshooting often requires a detailed analysis that may involve network packet capturing and a manual analysis using tools such as Wireshark. This is time-consuming and requires deep knowledge of communication protocols. Therefore, this domain is a suitable candidate for the deployment of an expert system. In this paper, we consider a rule-based system integrating the expert knowledge that performs an automatic root cause analysis of network problems identifiable from network communications. The system is open, thus it is possible to add new rules as needed, e.g., for specific and recurring cases of a target environment. The rules are evaluated in a tree-based fashion, which enables us to collect additional information during the problem search to better explain the possible causes. We successfully deployed the tool as part of a commercial tool for network monitoring.**

Keywords–Network diagnostics; rule-based diagnostics; fault tree analysis; event-based diagnostics; decision trees.

I. Introduction

Network infrastructure and applications are complex, prone to cyber attacks, outages, performance problems, misconfigurations, and problems caused by software or hardware incompatibility. All these problems may affect network performance and user experience [2], which may have fatal consequences for critical network infrastructure, e.g., e-health, e-government, Industrial IoT, smart grid, etc. Network troubleshooting is thus among the most common and important activities by network administrators. Despite the help of the current network monitoring tools, identification of a root cause of issues can be a complicated and mostly manual activity. The tools often reveal symptoms of the problem but the reasoning and problem localization are left for human operators expecting that they understand the problem and have sufficient knowledge of the technologies involved. Even if it is the case, the troubleshooting can be a lengthy and tiresome process that requires inspection of different sources of information, e.g., log files, the content of various tables, communication traces, etc. Application communication protocols are designed to implement the data exchange of remote parties. The protocol specification defines the syntax and meaning of messages, the way the conversation is controlled, and also the indication of error states. Thus, by inspecting the network communication it is possible to understand the situation and identify the indicated errors and in many cases also their probable cause.

Unfortunately, many network administrators do not have the proper tools and/or knowledge to diagnose and fix network problems effectively, and they require an automated tool to diagnose these errors [3]. Zeng et al. [4] provide a short survey on network troubleshooting from the administrators' viewpoint identifying the most common network problems: *reachability problems, degraded throughput, high latency, and intermittent connectivity*. The consulted network administrators expressed the need for a network monitoring tool that would be able to identify such problems.

This paper proposes a system, which creates diagnostic information only by performing passive network traffic packet-level analysis. Previous research and development provided tools for helping administrators to diagnose faults [5] and performance problems [6]. However, these tools either require *installation of agents on hosts, active monitoring, or providing rich information about the environment*. The idea behind our proposal is to automate the reasoning usually done by network analysts when investigating the root cause of an error from the captured network traces. It means that it is not necessary to change the network environment nor deploy any new devices. The troubleshooting process may remain unchanged except that one of the most labor-intensive parts represented by the packet-level traffic analysis is automated. Still, the user can verify the results obtained from the automated analysis as the process provides sufficient diagnostic information for the identification of problem relevant artifacts.

One of the most common ways of analyzing network traffic is by using a network packet analyzer (e.g., Wireshark). The analyzer works with captured network traffic (PCAP files) and displays structured information of layered protocols contained in every packet (encapsulated protocols, protocol fields). Administrators work with this information, check transferred content and compare the data with expected values. This process, done manually, is time-consuming and requires a good knowledge of network protocols and technologies.

The main contribution of this paper is a proposal of a tool for automatic diagnoses of network related problems from network communication only. Our approach tries to imitate a diagnostic process of a real administrator using the fault tree method and a popular packet parsing tool TShark. We have also implemented a proof-of-concept implementation to confirm the viability of the approach. This paper is an extension of our previous paper [1]. The most significant change is the improvement of input data processing. A new more efficient mechanism of converting input data into a specific indexable

format has been implemented. This change required significant modification of the method the system uses to access the data. However, the new format simplifies processing of other data types and reduces the execution time of the whole diagnostic significantly. A simple example of a tool usage for another data type (log files) is also presented.

The paper is organized as follows. Section II defines the problem statement and research questions. Section III discusses related work and describes diagnostic approaches. Our solution consists of five stages and is introduced in Section IV. Section V instructs network administrators how to use our system and shows how we model diagnostic knowledge. Section VI shows the output from the tool and evaluates the performance. Finally, Section VII is the conclusion, which summarizes the current state and proposes future work.

II. Research Questions

Our primary goal is to design a system that infers possible causes accountable for network related problems, such as service unreachability or application errors. Offering a list of actions for fixing the errors' cause is the secondary and optional goal. All this information is gathered only from captured network communication, which makes this approach applicable to various scenarios.

In our work, we focus on enterprise networks that have complex networking topologies, usually consisting of various network and end-point devices. The availability of network traces in the form of packet captures is essential to our method. Thus, we expect that administrators can collect network communication at appropriate locations in the network. Also, we consider that the capturing process creates packet captures without packet losses. As this may be difficult to guarantee for high-speed networks without using specialized hardware, for the diagnostic we usually do not require all communication. Thus, the packet capture can be recorded by applying a suitable filter to reduce the amount of data that needs to be processed.

To achieve our goal, we need to find answers to the following research questions:

1) How to model different network faults in a suitable way for implementation in a diagnostic system? *Reachability, application specific, and device malfunctioning problems* can cause various networking issues. We need to have a unified approach for modeling these problems to identify the symptoms and link them with root causes.
2) What information should be extracted from the captured network communication to identify symptoms of failures? In our case, we can passively access the communication in the monitored network and extract the necessary data to detect possible symptoms. An approach that can efficiently detect the symptoms in terms of precision and performance is needed.
3) How to identify the root cause of the problem, if we have a set of related symptoms? The core part of the diagnostic engine is to apply knowledge gathered from observed symptoms to infer the possible root cause of the problem. The result should provide the information in sufficient detail. For instance, if the authentication during the establishing of the connection fails, then we would like to know this specific information instead of a more general explanation (e.g., unable to establish a connection).
4) What actions can be provided to the administrator to fix the problems? Based on the observed symptoms and the root cause, the system should be able to provide fixing guidelines. These guidelines are supposed to be easy to understand even for an inexperienced administrator.

III. Related Work

A lot of research activities were dedicated to the diagnoses of network faults. Various methods were proposed for different network environments [5], in particular, home networks [7], enterprise networks [8]–[11], data centers [6], backbone and telecommunications networks [12], mobile networks [13], Internet of Things [14], Internet routing [15] and host reachability. Methods of network troubleshooting can be roughly divided into the following classes:

Active methods use traffic generators to send probe packets that can detect the availability of services or check the status of applications [16]. Usually, generators create diagnostic communication according to the test plan [8]. The responses are evaluated and provide diagnostic information that may help to reveal device misconfiguration or transient fail network states. Diagnostic probes introduce extra traffic, which may pose a problem for large installations [11]. Also, active methods may rely on the deployment of an agent within the environment to get information about the individual nodes [9].

Passive methods detect symptoms from existing data sources, e.g., traffic metadata [12], traffic capture files, network log files [15], performance counters. Passive methods can utilize the data commonly provided by various network monitoring systems.

Some systems combine passive traffic monitoring to detect faults with active probing to determine the cause of failure. Identifying anomalies related to network faults and linking them with possible causes commonly utilizes some of the following approaches:

Inference-based approach uses a *model* to identify the dependence among components and to infer the faults using a collection of facts about the individual components [9], [17].

Rule-based approach uses *predefined rules* to diagnose faults [10]. The rules identify symptoms and determine how these contribute to the cause. The rules may be organized in a collaborative environment for sharing knowledge between administrators [7]. Kim et al. [18] propose a rule-based reasoning (RBR) expert system for network fault and security diagnosis. The system uses a set of agents that provide facts to the diagnostics engine. De Paola et al. [19] deals with a distributed multi-agent architecture for network management. The implemented logical inference system enables automated isolation, diagnosis, and repairing network anomalies through the use of agents running on network devices. Dong and Dulay [20] developed an assumption-based argumentation to create an open framework of the diagnosis procedures able to identify the typical errors in home networks. Rule-based systems often do not directly learn from experience. They are also unable to deal with new previously unseen situations, and it is hard to maintain the represented knowledge consistently [5].

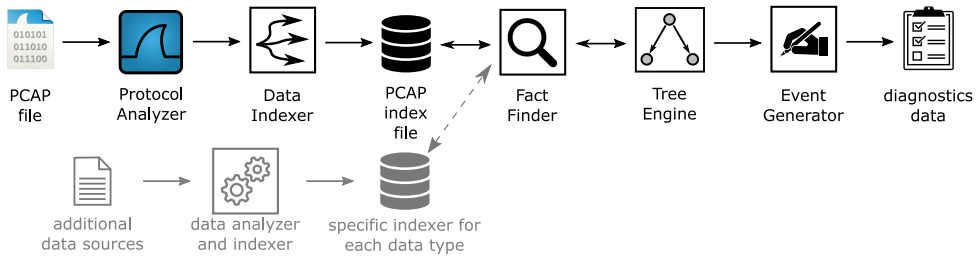

Figure 1. The top-level architecture of the proposed system. The architecture consists of five stages and one intermediate data storage (index file). The grey area represents optional architecture extensions — additional data sources.

Classifier-based approach *requires training data* to learn the normal and faulty states. The classifier can identify a fault and its likely cause [21]. Classifier-based methods were considered for misconfiguration detection in the home networks [22] and in the large network infrastructure [23]. *Tranalyzer* [24] is a flow-based analyzer that does traffic mining and a statistical analysis for large-scale networks. Big-DAMA [25] is a novel framework for detection and diagnosis of network traffic anomalies.

Network diagnostics based on traffic analysis can also use methods proposed for anomaly detection as some types of faults result in network communication anomalies.

Compared to other rule-based solutions, our system uses decision trees, which allows us to define more complex situations. Compared to simple rules (as used, for example, by a fishbone diagram), it is possible to make decisions based on previous diagnostic steps. Another difference is that our system does not need to know in advance what is wrong or what to focus on. Also, our system is not limited to only one type of data, and diagnostic rules are understandable by real administrators (not just scientists and programmers).

IV. Proposed System Architecture

We have built an expert system for analyzing network traffic, that has already been integrated into a worldwide business product [26]. The system combines rule-based and inference-based methods as it is easy to understand for network administrators. While the use of classifier-based methods has been proven very suitable for anomaly detection it lacks the capability to provide additional information for the detected case. The advantage of learning from provided data can only be exploited if a large set of annotated data is available. Contrary, the rule-based method can be extended also for detecting rare cases. The system only requires captured network traffic containing enough information about the event. Thus it is a completely passive method. Active methods generate additional traffic into a network (which can be unwanted in some situations) and require access to the network.

The proposed system is a processing pipeline that consists of several stages, as shown in Figure 1. The first stage, labeled as *Protocol Analyzer*, filters and decodes input packets using an external tool. The second stage takes decoded packets and converts them into a format for easier and faster data access (PCAP index file). The third stage, named *Fact Finder*, executes simple rules to identify facts significant from the diagnostics point of view. In the fourth *Tree Engine* stage, the decision tree utilizes the *Fact Finder* and identifies the possible problem cause. The fifth, and the last, stage *Event Generator* generates diagnostic outputs that contain detected errors and suggested solutions. Stages three, four, and five are easily extendable by the administrator who can add new rules and definitions.

The system can also be extended to use different data sources (e.g., log files or NetFlow records), as shown on the second row in Figure 1. Each data source requires specific data preprocessing that leads to the creation of an index file. The common part starts with the *Fact Finder* that can search indexed data of different data sources. If not specified otherwise, in the rest of the paper, we describe and evaluate the system only for a single data source represented by captured packet traces.

A. Protocol Analyzer

The first step in the processing pipeline is decoding captured network traffic in the PCAP format into a readable JSON format. We employ the tool TShark, which is a command-line version of the widely-used network protocol analyzer Wireshark. Because TShark follows the field naming convention used by Wireshark, we can use Wireshark Display Filter Expressions to select packet attributes. TShark supports all packet dissectors available in Wireshark. An example of TShark's output format with some omitted data is displayed in Figure 2.

```
{
...  "_source": {
       "layers": {
         "frame": {
           "frame.number": "15",
           "frame.len": "84",
...
         "ip": {
           "ip.ttl": "50",
           "ip.proto": "6",
...
         "tcp": {
           "tcp.srcport": "25",
           "tcp.dstport": "1470",
...
         "smtp": {
           "smtp.response.code": "235",
...
}
```

Figure 2. An output from the TShark's JSON format.

Using TShark brings the following benefits:

- many protocol dissectors are available and the community quickly provides a parser for an emerged protocol;
- tunneled, segmented and reassembled data are support;
- data presentation is consistent with the Wireshark, which allows the creation of an easy-to-read API for diagnostics.

TShark provides not only data of fields from supported network protocols but also some computed data, such as round trip time, missing or retransmitted TCP segments, which can be used in diagnostic rules.

Even if our primary use case is to diagnose problems inside the captured network data, we would like to test that our system can work with other data sources as well. For this test, we have chosen to use the log files. Because each application has its own format of log messages and we were not able to find a universal tool that can parse the content of any log message into a JSON object, we have implemented a custom parser.

Our data preparation script takes log records one by one, and if a record matches some of the predefined regular expressions, the record is converted into a JSON format, as shown in Figure 3. Currently, only a few applications are supported - postfix, dovecot, and fail2ban. The output JSON format has the same structure as the JSON from the TShark tool, so future processing will remain the same.

```
Feb 20 01:12:19 mail dovecot: auth: passwd-file(info,
185.36.81.57): unknown user (SHA1 of given password:
ece4e6)
```

```
{
   "time": "1582161139",      # Feb 20 01:12:19
   "service": "mail dovecot",
   "mode": "auth",
   "username": "info",
   "ip": "185.36.81.57",
   "description": "unknown user"
}
```

Figure 3. Conversion of a single log record into a JSON object.

B. Data Indexer

Data Indexer converts data from the JSON format into a format suitable for fast searching by packets' field names (attributes) and their values. Most of the time, it will not be necessary to process the packets one by one, which significantly improves the resulting diagnostic speed. Each input packet is indexed and the following data is stored:

1) the packet itself;
2) a set of all field names of the packet;
3) map of values assigned to each packet's field.

Figure 4 shows an example of how indexing works. The entire index is represented by an associative array. First, the packet is stored under the *_raw* key and a packet number (in this case, 3). Using this value, it is possible to retrieve the packet in the same format as returned by TShark. Subsequently, the packet number is stored under a set of all indexed packets stored under the *_packets* key. This set will simplify some operations, and its usage can be seen in Figure 5. In the next steps, for each packet attribute and its values (each attribute can contain multiple values) a set of packet numbers is created (when it does not already exist). After that, the current packet number is added to the set.

```
1 index = dict() # associative array in
      Python
2 index["_raw/3"] = {"frame.number":
      ["3"], "dns.id": ["0x00007df5"],
      "ip.addr": ["192.168.1.1",
      "192.168.1.100"],...} # under the
      _raw + packet number key, the
      original packet in the JSON format
      is stored
3 index["_packets"] = {1, 2, 3} #
      _packets index contains set of all
      packet numbers
4 index["dns.id"] = {3} # all packets
      with any "dns.id" field value are
      saved under the field name key
5 index["dns.id/0x00007df5"] = {3} #
      packets which contain "dns.id" field
       with value "0x00007df5" are saved
      under field name/field value key
6 index["ip.addr"] = {1, 2, 3}
7 index["ip.addr/192.168.1.1"] = {1, 3}
8 index["ip.addr/192.168.1.100"] = {3}
```

Figure 4. An example of the indexing of a few fields from the DNS packet. The bold text is showing index keys and values, which have been added because of the new packet.

C. Fact Finder

The *Fact Finder* aims to identify specific situations useful for network diagnostics. Facts can be attributed to one or more packets, which are in some relation. For example, a successful DNS name resolution is a fact that consists of a query and a corresponding reply DNS messages. The facts are specified by rules describing which packets should be found and which relation they should fulfill. The format of these rules is described in Subsection V-B. A rule can consist of up to three parts:

1) a list of packet filters;
2) a list of assertions to express relation constraints;
3) parameters for the filters and assertions.

The system evaluates rules as follows: (i) Parameters are replaced by provided values. (ii) Each packet filter returns a list of packets matching the filter. (iii) Assertions are evaluated to select sets of packets satisfying the constraints. A result has the form of a collection of sets of packet numbers, e.g., a rule that identifies DNS request-reply pairs checks that the transaction ID in both the request and reply packets match. The last step is converting sets of packet numbers into lists of packets. Packets in lists are ordered by the packet numbers.

Filter expressions use Wireshark's display filter language. By using this language, the expression can be first tested in Wireshark before it is used in a *Fact Finder* rule. Assertion constraints use our created language that is based on the Wireshark's display filter language. There are three changes made to the original language, which add support of:

1) working with packets from filter expressions;
2) simple math operations (+, -, *, /);
3) parameters for expressions. The parameters do not increase language capability but aim to simplify rule definition.

The evaluation of the facts begins with searching for packets with specific attributes. However, this varies depending on how these attributes are specified. In the case of a simple condition, it is possible to use the index created in the *Data Indexer* step, but with more complex conditions, this is not possible. A more complex condition is one that contains either a regular expression, function (string length, substring), or some comparison ($<, <=, >, >=$).

If it is possible to search for packets using the created index, the appropriate packet numbers are searched for using each attribute specified. Based on the specific relation between attributes, the adequate set operation is applied to the sets of packets. This process is shown in Figure 5. This figure describes finding packets by using the created index.

```
1  search: dns
2  result = index["dns"]
3
4  search: dns.flags.response == 1
5  result = index["dns.flags.response/1"]
6
7  search: dns and ip.addr == "10.10.10.1"
8  dns_packets = index["dns"]
9  ip_packets=index["ip.addr/10.10.10.1"]
10 result = dns_packets.intersection(
       ip_packets) # packets both in
       dns_packets and ip_packets
11
12 search:smtp.response.code != 250
13 all_packets = index["_packets"]
14 skip_packets = index["smtp.response.
       code/250"]
15 result = all_packets.difference(
       skip_packets) # packets in
       all_packets but not in skip_packets
```

Figure 5. An example of index usage when searching for packets that meet the specified constraints. When combining attributes in constraints, results from individual attributes are combined using set operations. The bold text shows the packet specification.

In case the packet specification cannot be evaluated using the created index, it is necessary to go through each packet and evaluate the condition for its values. This is accomplished by replacing the attributes in the expression (e.g., *smtp.response.code matches "[45] [0-9] [0-9]" and ip.addr = "10.10.1.1"*) with values from each packet. Because a list of values represents each attribute's value, the evaluation process must try all value combinations. If at least one combination fulfills the packet specification, the packet is added to the set of fulfilling packets. The principle is shown in Figure 6. Because there was not such a complicated rule in DNS protocol, we are showing this principle on the SMTP rule.

After all packets have been searched, they are represented only by a list of packet numbers. Before working with the packet's data, it is necessary to replace these packet numbers with the actual packets. After that, constraints defining packet relationships can be evaluated (assert rules). An example of a relationship is a request-reply pair of packets that are linked together by a request ID. Searching for such packets is accomplished by creating all possible packet combinations (Cartesian product) and evaluating all conditions for each combination.

```
1  search packets: smtp.response.code
       matches "[45] [0-9] [0-9]" and
       ip.addr = "10.10.1.1"
2  import re # regular expression module
3  result = set()
4  def check_packet(packet):
5    values = {}
6    for value in packet["smtp.response.
         code"]: # only packets with field
         smtp.response.code are used
7      values["smtp.response.code"]=value
8      for value in packet["ip.addr"]: #
           only packets with attribute ip.
           addr are used
9        values["ip.addr"] = value
10       if re.search(values["smtp.
             response.code"],"[45][0-9]
             [0-9]") and values["ip.addr"]
             == "10.10.1.1":
11         result.add(packet_number)
12         return result
13
14 for packet_number in index["_packets"]:
15   packet = index["_raw/"+packet_number]
16   result = check_packet(packet, result)
17 return result
```

Figure 6. An example of finding all packets that meet the specified condition, which can not be evaluated by using the created indexes. When evaluating a condition, all value combinations are tested for each packet. The bold text shows the packet specification and the corresponding condition.

The principle of evaluating the assert conditions is shown in Figure 7. The code in the figure contains a $relation()$ function that combines all possible values (similar to the code in Figure 6) and compares whether at least one value combination meets the defined relation function (e.g., " $==$ " for equality). The $relation()$ function works with a $packets$ dictionary that contains a list of packets that are saved under the keys defined in the facts section of the rule.

```
1  facts:
2    dns_query: dns.flags.response == 0
3    dns_reply: dns.flags.response == 1
4  asserts:
5   - dns_query[udp.stream] ==
      dns_reply[udp.stream]
6   - dns_query[dns.id] ==
      dns_reply[dns.id]
7
8  result = []
9  for query in packets["dns_query"]:
10   for reply in packets["dns_reply"]:
11     if relation(query["udp.stream"],
         "==", reply["udp.stream"])
12 and relation(query["dns.id"],"==",
       reply["dns.id"]): # relation()
       function checks all combinations of
       values from two lists
13       result.append({"dns_query": query
           , "dns_reply": reply})
14 return result
```

Figure 7. Example of a packet set search (DNS query and response) that meets the defined constraint (packets from the same UDP stream and the same request ID). The bold text is sharing assert constraints and the corresponding $relation()$ function.

D. Tree Engine

The tree engine infers the possible error cause by evaluating a decision tree that contains expert knowledge about supported network protocols and services. Each node of the tree contains a diagnostic question. Questions refer to facts identified by the *Fact Finder*. Based on the question's result, the next tree node is chosen. This node transition creates a path that begins in a root node and finishes in a leaf node. Paths in the tree represent gathered knowledge and lead to the possible cause of the problem.

The decision tree consists of declarative specifications of tree nodes enriched by Python code. The declarative part is responsible for creating the tree and consists of a rule name, a rule type, a *Fact Finder* rule, and two branches, which cover the success and the fail result of the *Fact Finder*. Both branches can define the next rule, which should be processed.

Python codes are located inside the success and fail branches. These codes are responsible for processing logic (e.g., saving packets for future tree nodes or translating error codes from packets into human-readable format) and generating diagnostic results. The format of rules is described in Subsection V-A.

E. Event Generator

During the diagnostic process, a report is created to provide diagnostic information for network administrators. The diagnostic report is produced in a human-readable format, as well as in a machine format useful for further processing or visualization. The report consists of events that are constructed in tree nodes based on the derived knowledge and processed packets. Each event describes one situation that happened in the network. For example, the connection to the HTTP server has been detected.

Each rule consists of a name, description, and severity of the detected situation. Additionally, the event may include a suggestion message and data from the provided packet. The provided packet is specified as a parameter in the tree rule. By using this packet, parameters such as flow identification or timestamp can be associated with the event. Subsection V-C describes the format of the event rules.

V. Rule Specification

The diagnostic engine defines each protocol as a decision tree. The tree consists of nodes representing administrator questions, and edges representing answers to these questions. The edge can move the diagnostic process from one question to another (within the same protocol or another) or finish the process with the discovered result.

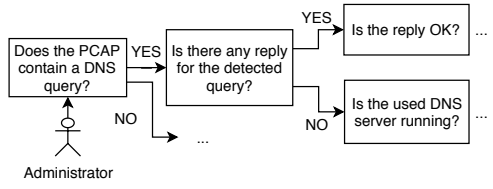

Figure 8. A simple illustration of a binary decision tree. An administrator diagnoses a DNS problem by anwering questions in the predefined order.

The questions simulate thinking of a real administrator. Typically, an administrator starts to search for certain network packet values and after the search for them is finished, the administrator searches for next values based on the result. In our solution, each question can only have two answers: success or failure. This yields a binary decision tree. Figure 8 shows an example of a small portion of the DNS tree.

The decision tree needs to be converted to a format understandable by our system. This conversion is split into three steps: 1) defining tree nodes (*Tree node* rules), 2) defining conditions for choosing tree nodes (*Fact Finder* rules) and 3) defining the diagnostic report (*Event definition* rules). The following subsections describe the syntax for each of these rules. The reason why a node rule does not contain a lookup code and an event definition directly and they need to be defined in separate rules is that multiple rules would not be able to use the same lookup code and events (increases reusability).

The conversion of the decision tree assigns a name to each tree node. We use the node names as labels for switching from one node to another. Each node tries to find specific facts, defined as a *Fact Finder rule*. Based on the condition, if some fact was found or not, the next diagnostic step is chosen. Each rule can have one or none success and fail branches. Branches contain executable Python code and the next node rule name. After the execution of the Python code, the analysis switches to the next node. Figure 9 shows the pseudocode for writing tree nodes.

```
1 tree_node_id:
2   if fact_finder_rule finds some facts:
3     success branch_code
4     jump to the next tree_node_id
5   else:
6     fail branch_code
7     jump to the next tree_node_id
```

Figure 9. Pseudocode for writing a tree node. Each node should have a unique id, lookup condition, and branch codes.

A. Tree Node Rules

All the rules are saved in a declarative YAML format. This format is easily understandable by programming code and by people without programming skills (we assume that not all network administrators are also programmers). Even if the system already contains some protocols, the administrators can easily add new protocols or can extend capabilities of the current protocols by updating the rules. In the following paragraphs, the format of the rules will be described. Names of the sections as they used in rules are placed inside the text.

The rule definition begins with the rule name (rule section *id*) and the execution of a *Fact Finder* rule (rule section *query*). The result of the *Fact Finder* is a list of associative arrays. Each array can contain multiple packets, where the packet name is the key to the array. These packets will be processed according to the *Tree Node* rule type (rule section *type*). The default behavior selects the first array from the list of arrays (the list is ordered by the arrival time of packets) and marks it as a found fact. The second type of rule is a "foreach" type, which iterates through the list of arrays and progressively marks each array as a fact and executes the defined rule. For example, the foreach rule can analyze each query to the server or each response to the selected query (as shown in Figure 10).

```
id: DNS query detected # name of the rule
query: exists DNS reply for the detected query? # Facts Finder rule
type: foreach
success:
 state: is reply ok? # next state
 code: | # Python code follows
  reply_pkt = _fact["dns_reply"]
  save("dns_reply", reply_pkt)
  event("reply_detected", reply_pkt)
fail:
 state: find any reply from the same destination server # next state
 code: | # Python code follows
  query_pkt = load("dns_query")
  event("reply_not_detected",query_pkt)
```

Figure 10. Simple Tree Engine rule showing what should be done if a DNS query was detected.

Furthermore, the rule consists of two parts, with only one executed (depending on whether the diagnostic engine has found the searched fact or not). The format of both parts is the same. Each part consists of the name of the next rule with which the diagnostics should continue (rule section *state*) and the Python code (rule section *code*). Each rule can switch to to a rule from another protocol to diagnose problems across several protocols, e.g., if an SMTP communication is not detected, we will check if there are any ICMP unreachable messages, failed TCP connection attempts or incorrect DNS resolutions. If the next rule is not specified, the diagnostic engine stops the diagnostic process.

The Python code can process packet data, make logical decisions and most importantly, generate diagnostic output. Within the Python code, it is possible to use any Python 3 code and it is also possible to utilize the following variables and functions defined by the engine:

1) *fact* - contains the first fact found (or the next one in the foreach type rule)
2) *facts* - contains all the facts found
3) *save()* - saving any value for further processing (inside another Tree node rule or as a parameter in the *Fact Finder* rule);
4) *load()* - read the value previously saved by the save() function;
5) *event()* - generates a diagnostic report, where the parameter is a packet to which the report refers.

Figure 10 shows an example of a rule defining the middle node from the tree in Figure 8. The figure shows a node describing that a DNS query has been detected (*id*) and the rule is looking for a DNS response for the detected query (*query*). For each detected reply, a successful section (*success*) is executed (foreach *type*). The response is saved, the diagnostic message is generated (*code*), and the diagnostic process continues to check whether the response is without error (*state*). If no response to the query is found, the failure section is executed (*fail*). First, the original query is retrieved, the diagnostic message is generated (*code*), and then the diagnostic process continues with the next state (*state*).

B. Fact Finder Rules

Rules in this section describe how a question is converted into packet lookup functions. Each rule may look for several independent packets, which are combined and checked if their relation fulfills assert conditions. Each question returns a list of associative arrays, where each arrays represents a unique combination of packets fulfilling the assert conditions (packet names are arrays keys).

Each rule needs to have a name (rule section *id*), which is used inside the *Tree Node* rules. The rule can have parameters to make rules more reusable (rule section *params*). For example, instead of creating a rule for each error code, it is possible to create one rule with a parameter containing the expected error code. Parameters are used as variables in the following sections and can have a format of a single value or single packet (previously saved by *save()* function within the *Tree Node* rule). Rule section *facts* define the name of the searched packets and the filter. The filter uses Wireshark display language and specifies which packets should be assigned to the specified name (each packet can be assigned to multiple packet names). Rule section *asserts* define conditions for the detection of packets. The conditions use our custom language, which is based on the Wireshark display language. However, the custom language allows to:

- use math operations addition(+), subtraction(-), multiplication(*) and division(/);
- use single value parameters as if the values were directly inserted into the condition;
- use values from packets (provided from packets or params section). The format is packet_name[field.name], where field.name is the Wireshark name assigned to the attribute.

Figure 11 shows an example of a simple rule for the question *Is there any DNS reply for the detected DNS query?* After the rule name (*id*), the parameter *dns_query* for the assert conditions is specified (*params*). The rule contains a definition of the *dns_reply* packet (*packets*), which is used in the assert conditions. The conditions (*asserts*) are checking whether the *dns_reply* belongs to the same UDP stream as the provided *dns_query* and if the reply packet is answering to the specific query.

```
id: exists DNS reply for the detected query?  # name of the rule
params: # saved data from any tree rule
 -dns_query
facts: # which packets we are looking for
 -dns_reply: dns.flags.response == 1
asserts: # packets relation constrain
 -dns_query[udp.stream]==dns_reply[udp.stream]
 -dns_query[dns.id]==dns_reply[dns.id]
```

Figure 11. Example of a DNS fact rule for checking if the PCAP file contains a reply for the provided query or not.

By default, the *Fact Finder* rules are working with the data saved inside the PCAP index file. To allow searching for facts from different data index files, it is necessary to specify the data type (rule section *type*). Figure 12 shows an example of such a rule that looks for a log record containing specific values related to the wrong username event. Because the rules are

using the same *Tree Node* engine, the rule of one type can use parameters from a different type of rule. In our example, we are looking for a record with the same IP address as used in the provided *imap_query* packet.

```
id: wrong username?
params: # saved packet from previous
    rules
 -imap_query
type: log # the data are saved in
    different index file
facts:
 -auth: description == "unknown user"
     and ip == imap_query[src.ip]
```

Figure 12. Fact rule for different data source (log file). The rule is checking presence of a specific record in input log file.

C. Event Definitions

Event rules describe how the diagnostic message will look. The message is created by calling a function *event()* from the *Tree Node* rule. In addition to the event name, the event function also accepts a packet parameter (the event is related to this packet). The idea is that from the provided packet, the time, the flow identification, and possibly other specified values are extracted and inserted into the message.

Each rule consists of a name, severity, description, instructions on how to fix the problem, and a list of fields from the provided packet. The field list contains the names according to the Wireshark terminology. An optional part of each field is its description to help the administrator understand its value. The description and suggestion may contain variables. The variables are written as $\{fieldname\}$, and will be replaced by values from the provided packet when the diagnostic report is generated.

Figure 13 shows an event describing the error that no DNS response was detected for the DNS query. From the provided packet, the queried domain name is inserted into the description and the DNS server address into the suggestion. Additional items will be also included in the output: transaction ID, queried domain name, and DNS server IP address.

```
id: reply_not_detected
severity: error
description: "No reply for query '{dns.
    qry.name}' has been detected."
suggestion: "Check if the DNS service is
     running on {ip.dst}. If yes, check
    the firewall on the server and the
    path between server and the client."
fields:
- name: dns.id
  description: Transaction ID
- name: dns.qry.name
  description: Queried domain name
- name: ip.dst
  description: Server IP address
```

Figure 13. A DNS event example, that reports that the query wasn't detected.

VI. Use Case and Evalution

The goal of the tool is on-demand diagnostic of the selected network traffic. It is essential to note that the goal is not an on-line (24/7) analysis or analysis of a large amount of data. The idea is based on a use case that when an administrator detects a problem on the network, it triggers a capture on the selected network traffic. For example, if a client with the address 192.168.0.20 is unable to establish a TLS connection with the server on the address 192.168.0.1, the administrator will capture the communication between these two stations.

We have implemented diagnostic rules for several application and service protocols. Table I shows the current list of supported protocols and their complexity in term of *Tree node* and *Fact Finder* rule count, and their capabilities in term of *Event* rule count.

Table I. Supported protocols and amount of rules and success, warning, error events which describe various protocol behavior situations.

Protocol	Tree nodes	Fact finders	Events		
			Success	Warning	Error
DHCP	24	22	10	9	4
DNS	12	12	8	4	5
FTP	24	10	15	6	7
HTTP	3	3	2	1	1
ICMP	4	2	0	0	4
IMAP	15	8	7	3	9
POP	21	7	5	10	7
SIP	38	22	15	1	8
SLAAC	8	7	1	6	1
SMB	27	25	20	3	5
SMTP	17	13	9	6	9
SSL	2	2	2	0	1
TCP	10	10	0	7	2

We have tested the functionality and performance of the implemented tool. Table II provides a sample of data from performance experiments. The execution time is divided into TShark tool processing time, time used for indexing the JSON from TShark, and the analysis time. Five files of different sizes containing some representative data are presented. The tests were performed on a CPU Intel Xeon Silver 4116 2.10 GHz and 4 GB RAM. It should be noted that only one CPU core was used for the diagnostics on the given processor, as TShark as well as the implemented tool are single-core applications.

Table II. The table shows the diagnostics execution time for the selected PCAP files of different sizes.

Size [MB]	Packets	Flows	Time [s]			
			TShark	Indexes	Analysis	Total
182.253	222 372	7155	18.717	27.407	19.678	65.802
21.569	62 471	7002	5.004	5.940	7.748	18.692
9.640	14 509	954	1.969	1.533	1.464	4.966
1.687	3 544	373	1.295	0.589	1.186	3.070
0.978	4 848	84	0.821	0.669	1.291	2.781

The output of the tool is a report in JSON format, which enables easy machine processing. We have created a web interface to visualize the report in a more human-readable format. The visualization consists of two parts. The first part shows a list of all detected events. After clicking on any event, the detail of this event is displayed in the second part. Below an event name, a suggestion for fixing the problem is displayed in which real values from the packet have replaced the message variables (written in curly brackets). After the suggestion message, the rest of the event attributes are displayed.

To demonstrate the functionality of the tool, we have diagnosed a PCAP file that was captured on a station with the IP address 10.10.1.4. The tool diagnoses all predefined protocols and displays the detected events in a hierarchical structure. Figure 14 shows this situation, with some events omitted for simplicity. For each event, a description and an icon of the event are displayed. The highest severity is then propagated from the deepest events out so that it is possible to find problem situations in a large number of events quickly. In addition to the detected error with the DNS response, it is possible to see other communications in the picture, which were without error.

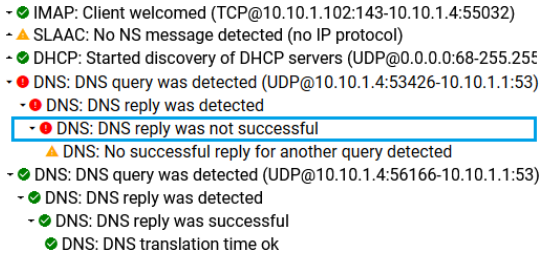

Figure 14. The figure shows a list of detected events for a diagnosed PCAP file. The list contains events from multiple protocols with different severities. The screenshot was taken from the *Flowmon Packet Investigator*, a product that has integrated the proposed tool.

After selecting an event from the list (represented by a blue rectangle), a detailed description of the event is displayed. Figure 15 shows this output, which describes the reason why the domain name translation failed. In addition to the event name, the listing is divided into three sections: 1) suggestions for the administrator on how to fix the error, 2) a brief summary of the event (description, severity, and flow), and 3) attributes extracted from the packet that triggered the displayed event.

DNS reply was not successful

Check if the domain 'naps.google.com' is correctly specified (e.g., some typo error) and check the DNS server '10.10.1.1' configuration.

Description	Failure reply with code '3' has been detected.
Protocol	DNS
Severity	error
Flow	UDP@10.10.1.1:53-10.10.1.4:53426
decoded error code	Domain name does not exist
Frame time epoch	12.11.2016 17:32:23
Frame number	2
IP version	4
IP source	10.10.1.1
IP destination	10.10.1.4
IP proto	17
UDP source port	53
UDP destination port	53426
dns.id	0x00000c01
dns.qry.name	naps.google.com
dns.flags.rcode	3

Figure 15. The figure shows an example of diagnostic output for a DNS error. It suggests to an administrator to check the domain name and the server configuration. The screenshot was taken from the *Flowmon Packet Investigator*, a product that has integrated the proposed tool.

VII. Conclusion

Network troubleshooting can be a nightmare for administrators because of system complexity. There may not be an evident link between the issue reported by a user and the real cause of the problem. Some of the errors can be identified and analyzed by examining network traffic. However, using the traditional mostly manual approach is time-consuming and requires significant expertise. The presented paper describes the automatized approach to network traffic analysis able to identify errors using the rule-based approach. The rules encode expert knowledge and are evaluated for the captured traffic.

While the rule-based approach may be considered as an old-fashioned approach these days when the majority of research considers a machine learning-based approach, it was demonstrated that knowledge encoded in the form of rules provides an efficient method for network troubleshooting. Moreover, because of expert-designed rules, it is possible to add information that explains the possible cause of the issue and recovery options. Mainly the explainability associated with this approach seems to be the biggest benefit for users. The drawback, of course, is related to the necessity of creating and testing the rules base. Also, the method is susceptible to the quality of data sources. When the captured communication is incomplete, the method can provide incorrect results.

The performance is important for any method to be practically usable. The presented method uses a rule evaluation engine that traverses decision trees for problem domains, which can be evaluated in a reasonable timeframe. However, as nodes of the tree contain expressions that can potentially require complex operations over the input data, the set of indexes is precomputed to improve the performance.

The proof-of-concept demonstrating the approach was implemented and further finalized to the tool integrated into the commercial suite for network monitoring. The tool is commercially provided on the market by Flowmon Networks company as Flowmon Packet Investigator [26].

Future work will focus on:

- adding support of new protocols, e.g., NTP or SNMP;
- even though the current performance is good enough, it can always be better, and our goal will be to decrease the execution time of the diagnostic process;
- because the quality of the diagnostic output highly depends on the quality of the input data, we would like to create a validation technique (maybe by using machine learning techniques) to check the validity of the input data (e.g., detection of packets loss);
- after separating the processing of the input data from the Fact Finder into Data Indexer, it is now possible to create a distributive solution that consists of many data collection points across the network. At each point, the data would be indexed by the Data Indexer and sent to the central processing unit.

Acknowledgment

This work was supported by project ”Network Diagnostics from Intercepted Communication” (2017-2019), no. TH02010186, funded by the Technology Agency of the Czech Republic, the BUT FIT grant FIT-S-20-6293, ”Application of AI methods to cyber se-curity and control systems”, and by private network monitoring company Flowmon Networks.

REFERENCES

[1] M. Holkovič and O. Ryšavý, "Network diagnostics using passive network monitoring and packet analysis," The Fifteenth International Conference on Networking and Services (ICNS), 2019, pp. 47–51.

[2] R. Wang, D. Wu, Y. Li, X. Yu, Z. Hui, and K. Long, "Knight's tour-based fast fault localization mechanism in mesh optical communication networks," Photonic Network Communications, vol. 23, no. 2, 2012, pp. 123–129.

[3] M. Solé, V. Muntés-Mulero, A. I. Rana, and G. Estrada, "Survey on models and techniques for root-cause analysis," arXiv preprint arXiv:1701.08546, 2017.

[4] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown, "A survey on network troubleshooting," Technical Report Stanford/TR12-HPNG-061012, Stanford University, Tech. Rep., 2012.

[5] M. łgorzata Steinder and A. S. Sethi, "A survey of fault localization techniques in computer networks," Science of computer programming, vol. 53, no. 2, 2004, pp. 165–194.

[6] C. Guo et al., "Pingmesh: A large-scale system for data center network latency measurement and analysis," in ACM SIGCOMM Computer Communication Review, vol. 45, no. 4. ACM, 2015, pp. 139–152.

[7] B. Agarwal, R. Bhagwan, T. Das, S. Eswaran, V. N. Padmanabhan, and G. M. Voelker, "Netprints: Diagnosing home network misconfigurations using shared knowledge," in Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, ser. NSDI'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 349–364.

[8] L. Lu, Z. Xu, W. Wang, and Y. Sun, "A new fault detection method for computer networks," Reliability Engineering & System Safety, vol. 114, 2013, pp. 45–51.

[9] S. Kandula et al., "Kandula, srikanth and mahajan, ratul and verkaik, patrick and agarwal, sharad and padhye, jitendra and bahl, paramvir," ACM SIGCOMM Computer Communication Review, vol. 39, no. 4, 2009, pp. 243–254.

[10] M. Luo, D. Zhang, G. Phua, L. Chen, and D. Wang, "An interactive rule based event management system for effective equipment troubleshooting," in IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society. IEEE, 2011, pp. 2329–2334.

[11] A. Mohamed, "Fault detection and identification in computer networks: A soft computing approach," Ph.D. dissertation, University of Waterloo, 2010.

[12] D. Brauckhoff, X. Dimitropoulos, A. Wagner, and K. Salamatian, "Anomaly extraction in backbone networks using association rules," in Proceedings of the 9th ACM SIGCOMM conference on Internet measurement. ACM, 2009, pp. 28–34.

[13] L. Benetazzo, C. Narduzzi, P. A. Pegoraro, and R. Tittoto, "Passive measurement tool for monitoring mobile packet network performances," IEEE transactions on instrumentation and measurement, vol. 55, no. 2, 2006, pp. 449–455.

[14] K.-H. Kim, H. Nam, J.-H. Park, and H. Schulzrinne, "Mot: a collaborative network troubleshooting platform for the internet of things," in Wireless Communications and Networking Conference (WCNC), 2014 IEEE. IEEE, 2014, pp. 3438–3443.

[15] T. Qiu, Z. Ge, D. Pei, J. Wang, and J. Xu, "What happened in my network: mining network events from router syslogs," in Proceedings of the 10th ACM SIGCOMM conference on Internet measurement. ACM, 2010, pp. 472–484.

[16] M. Vásquez-Bermúdez, J. Hidalgo, M. del Pilar Avilés-Vera, J. Sánchez-Cercado, and C. R. Antón-Cedeño, "Analysis of a network fault detection system to support decision making," in International Conference on Technologies and Innovation. Springer, 2017, pp. 72–83.

[17] S. Jamali and M. S. Garshasbi, "Fault localization algorithm in computer networks by employing a genetic algorithm," Journal of Experimental & Theoretical Artificial Intelligence, vol. 29, no. 1, 2017, pp. 157–174.

[18] S. Kim, S. j. Ahn, J. Chung, I. Hwang, S. Kim, M. No, and S. Sin, "A rule based approach to network fault and security diagnosis with agent collaboration," in Artificial Intelligence and Simulation, T. G. Kim, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 597–606.

[19] A. De Paola, S. Fiduccia, S. Gaglio, L. Gatani, G. Lo Re, A. Pizzitola, M. Ortolani, P. Storniolo, and A. Urso, "Rule based reasoning for network management," in Seventh International Workshop on Computer Architecture for Machine Perception (CAMP'05), July 2005, pp. 25–30.

[20] C. Dong and N. Dulay, "Argumentation-based fault diagnosis for home networks," in Proceedings of the 2nd ACM SIGCOMM Workshop on Home Networks, ser. HomeNets '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 37–42. [Online]. Available: https://doi.org/10.1145/2018567.2018576

[21] E. S. Ali and M. Darwish, "Diagnosing network faults using bayesian and case-based reasoning techniques," in Computer Engineering & Systems, 2007. ICCES'07. International Conference on. IEEE, 2007, pp. 145–150.

[22] B. Aggarwal, R. Bhagwan, T. Das, S. Eswaran, V. N. Padmanabhan, and G. M. Voelker, "NetPrints: Diagnosing home network misconfigurations using shared knowledge," Proceedings of the 6th USENIX symposium on Networked systems design and implementation, vol. Di, no. July, 2009, pp. 349–364. [Online]. Available: http://portal.acm.org/citation.cfm?id=1559001

[23] M. Chen, A. Zheng, J. Lloyd, M. Jordan, and E. Brewer, "Failure diagnosis using decision trees," International Conference on Autonomic Computing, 2004. Proceedings., 2004, pp. 36–43. [Online]. Available: http://ieeexplore.ieee.org/document/1301345/

[24] S. Burschka and B. Dupasquier, "Tranalyzer: Versatile high performance network traffic analyser," in 2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016, 2017.

[25] P. Casas, T. Zseby, and M. Mellia, "Big-DAMA: Big Data Analytics for Network Traffic Monitoring and Analysis," Proceedings of the 2016 Workshop on Fostering Latin-American Research in Data Communication Networks (ACM LANCOMM'16), 2016.

[26] "Flowmon products overview," https://www.flowmon.com/en/overview, accessed: 2020-May-27.

A.6 Application Error Detection in Networks by Protocol Behavior Model

Authors: Holkovič Martin, Ing. (70%), Polčák Libor, Ing., Ph.D. (15%), Ryšavý Ondřej, doc. Ing., Ph.D. (15%)
Abstract: The identification of causes of errors in network systems is difficult due to their inherent complexity. Network administrators usually rely on available information sources to analyze the current situation and identify possible problems. Even though they are able to identify the symptoms seen in the past and thus can apply their experience gathered from the solved cases the time needed to identify and correct the errors is considerable. The automation of the troubleshooting process is a way to reduce the time spent on individual cases. In this paper, the model that can be used to automate the diagnostic process of network communication is presented. The model is based on building the finite automaton to describe protocol behavior in various situations. The unknown communication is checked against the model to identify error states and associated descriptions of causes. The tool prototype was implemented in order to demonstrate the proposed method via a set of experiments.
Keywords: Network diagnostics, automatic diagnostics, timed automata, protocol model from traces, encrypted data diagnostics, application behavior model.
Published in: E-Business and Telecommunications
Conference rating: C (Core), B4 (Qualis)
ISSN: 1865-0929
ISBN: 978-3-030-52685-6

Application Error Detection in Networks by Protocol Behavior Model

Martin Holkovič[1], Libor Polčák[2][0000−0001−9177−3073], and Ondřej Ryšavý[2][0000−0001−9652−6418]

[1] Brno University of Technology, Faculty of Information Technology, NES@FIT, Bozetechova 1/2, 612 66 Brno, Czech Republic
[2] Brno University of Technology, Faculty of Information Technology, Centre of Excellence IT4Innovations, Bozetechova 1/2, 612 66 Brno, Czech Republic
{iholkovic,polcak,rysavy}@fit.vutbr.cz

Abstract. The identification of causes of errors in network systems is difficult due to their inherent complexity. Network administrators usually rely on available information sources to analyze the current situation and identify possible problems. Even though they are able to identify the symptoms seen in the past and thus can apply their experience gathered from the solved cases the time needed to identify and correct the errors is considerable. The automation of the troubleshooting process is a way to reduce the time spent on individual cases. In this paper, the model that can be used to automate the diagnostic process of network communication is presented. The model is based on building the finite automaton to describe protocol behavior in various situations. The unknown communication is checked against the model to identify error states and associated descriptions of causes. The tool prototype was implemented in order to demonstrate the proposed method via a set of experiments.

Keywords: Network diagnostics · automatic diagnostics · timed automata · protocol model from traces · encrypted data diagnostics · application behavior model.

1 Introduction

Computer networks are complex systems equipped with different network devices and hosts that provide and consume application services. Various types of errors, such as misconfiguration, device failures, network application crashes, or even user misbehavior can cause that expected network functions are not available. Users perceive network problems by the inaccessibility of web services, the degraded performance of network applications, etc. Usually, it is the role of network administrators to identify the cause of problems and to apply corrective activities in order to restore the network functions again.

The network troubleshooting process is often described as a systematic approach to identify, diagnose and resolve problems and issues within a computer network. Despite the published procedures, methods and techniques, and tool

support, the network diagnostics is a largely manual and time-consuming process. Troubleshooting often requires expert technical knowledge of network technologies, communication protocols, and network applications. Another complication is that the administrator often needs to check the number of possible sources to find the real source of the problem. It amounts to check log files in network devices or network applications, the current content of various tables, traces of network communication, etc. Although an experienced administrator usually has advanced skills in network troubleshooting that helps the administrator to quickly identify problems there may be situations that are hard to solve and not evident until the detailed network communication analysis is carried out.

The need for advanced tools that support network diagnostics is expressed by most network professionals surveyed in the report presented by Zeng et al. [31]. Existing tools can provide various information about the network, such as service status and performance characteristics, which is useful for problem detection but they often do not provide enough information for the cause identification. In computer networks, there can happen a lot of different problems. Many of them can be identified by using a network traffic analyzer. The traffic analyzer is a software to intercept the data packet flow that in the hand of an experienced administrator enables to check for the latency issues and other networking problems which help to reveal the root cause. However, using a traffic analyzer requires an understanding of different communication protocols. Also, the number of flows that need to be analyzed can be large making the analysis long and tedious task.

In order to improve the network troubleshooting process, we propose to develop a tool that automatically generates a protocol behavior model from the provided examples (traces) of the protocol conversations. In particular, a network administrator is required to provide two groups of files. The first group contains traces of normal (expected) behavior, while the second group consists of known, previously identified error traces. Based on these distinct groups, the tool is able to construct a protocol model that can be later used for detection and diagnosis of issues in the observed network communication. Once the model is created, additional traces may be used to improve the model gradually.

When designing the system, we assumed some practical considerations:

- It should not need to be required to implement custom application protocol dissectors to understand the communication.
- Application error diagnostics cannot be affected by lower protocols, e.g., version of IP protocol, data tunneling protocol.
- The The model should be easily interpretable and also useful for other activities too, e.g., security analysis, performance analysis.

The main benefit of this work is a new automatic diagnostic method for the detection of errors observable from the application protocol communication. The method is based on the construction of a protocol behavior model that contains both correct and error communication patterns. An administrator can also use the created model for documentation purposes and as part of a more detailed analysis, e.g., performance or security analysis.

This paper is an extended version of the paper "*Using Network Traces to Generate Models for Automatic Network Application Protocols Diagnostics*" [13]. We have added and improved several parts that extend the original paper, in particular: i) models of protocols are better described, ii) the processing of ungeneralizable requests (and states) have been completely reworked, iii) the diagnostics engine can now include time information, iv) and preliminary evaluation of encrypted traffic analysis was realized.

The focus of the previous contribution was only on detecting application layer errors in enterprise networks. We did not consider errors occurred on other layers and domains, e.g., wireless communication [24], routing errors [10], or performance issues [19]. However, in this extended version, we are also able to cope with performance problems. Because we are focusing on enterprise networks, we have made some assumptions on the accessiblity of data sources. For instance, we expect that administrators using this approach have full access to network traffic in the network. Even if the communication outside the company's network is encrypted, the traffic between the company's servers and inside the network can be sometimes available unencrypted, or the data can be decrypted by providing server's private key or logging the session keys [3]. However, because the encrypted traffic forms the majority of all communication on the Internet, we also preliminary evaluated whether the presented approach (generating models from traces) is applicable to encrypted traffic.

The paper is organized as follows: Section 2 describes existing work comparable to the presented approach. Section 3 defines model used for diagnostics. Section 4 overviews the system architecture. Section 5 provides details on the method, including algorithms used to create and use a protocol model. Section 6 presents the evaluation of the tool implementing the proposed system. Section 7 discusses some problems related to our approach. Finally, Section 8 summarizes the paper and identifies possible future work.

2 Related Work

Traditionally, error detection in network systems was mostly a manual process performed by network administrators as a reaction to the user reported or detected service unavailability or connectivity loss. As it is a tedious task various tools and automated methods were developed. A survey by [26] classifies the errors to network systems as either *application-related* or *network-related* problems. The most popular tool for manual network traffic analysis and troubleshooting is Wireshark [20]. It is equipped with a rich set of protocol dissectors that enables to view details on the communication at different network layers. However, an administrator has to manually analyze the traffic and decide which communication is abnormal, possibly contributing to the observed problem. Though Wireshark offers advanced filtering mechanism, it lacks any automation [12].

Network troubleshooting can be done using active, passive, or hybrid methods [27]. Active methods rely on the tools that generate probing packets to locate

[3] http://www.root9.net/2012/11/ssl-decryption-with-wireshark-private.html

network issues [2]. Specialized tools using generated diagnostic communication were also developed for testing network devices [23]. The advantage of active methods is that it is possible to detect a certain class of errors quickly and precisely identify the problem. On the other hand, generating diagnostic traffic may be unwanted in some situations. Passive detection methods rely on information that can be observed in network traffic or obtained from log files, dumps, etc.

During the course of research on passive network diagnostic methods, several approaches were proposed utilizing a variety of techniques. In the rest of this section, we present the different existing approaches to network diagnostics closely that relates to the presented contribution.

Rule-based Methods. Rule-based systems represent the application of artificial intelligence (reasoning) to the problem of system diagnosis. While this approach was mainly popular for automated fault detection of industrial systems, some authors applied this principle to develop network troubleshooting systems. [15] introduced rule-based reasoning (RBR) expert system for network fault and security diagnosis. The system uses a set of agents that provide facts to the diagnostics engine. [9] proposed distributed multi-agent architecture for network management. The implemented logical inference system enables automated isolation, diagnosis, and repairing network anomalies through the use of agents running on network devices. [11] employed assumption-based argumentation to create an open framework of the diagnosis procedures able to identify the typical errors in home networks. Rule-based systems often do not directly learn from experience. They are also unable to deal with new previously unseen situations, and it is hard to maintain the represented knowledge consistently [25].

Protocol Analysis. Automatic protocol analysis attempts to infer a model of normal communication from data samples. Often, the model has the form of a finite automaton representing the valid protocol communication. An automatic protocol reverse engineering that stores the communication patterns into regular expressions was suggested in [30]. Tool *ReverX* [3] automatically infers a specification of a protocol from network traces and generates corresponding automaton. Recently, reverse engineering of protocol specification only from recorded network traffic was proposed to infer protocol message formats as well as certain field semantics for binary protocols [17]. The automated inference of protocol specification (message format or even protocol behavior model) from traffic samples was considered by several authors. [8] presented *Discover*, a tool for automatic protocol reverse engineering of protocol message formats from network traces. The tool works for both binary and text protocols providing accuracy about 90%. The different approach to solve a similar goal was proposed by [29]. They instrumented network applications to observe the operation of processing network messages. Based on this information their method is able to recreate a message format, which is used to generate protocol parser. This work was extended by the same authors in [7] with an algorithm for extracting the state machine for the analyzed protocol. [16] developed a method

based on Markov models called *PRISMA*, which infers a functional state machine and message format of a protocol from network traffic alone. While focused on malware analysis, the tool is capable to identify communication behavior of arbitrary services using binary or textual protocols. Generating application-level specification from network traffic is addressed by [28]. They developed a system called *Veritas* that using the statistical analysis on the protocol formats is able to generate a probabilistic protocol state machine to represent the protocol flows.

Statistical and Machine Learning methods. Statistical and machine learning methods were considered for troubleshooting misconfigurations in the home networks by [1] and diagnosis of failures in the large networks by [6]. *Tranalyzer* [4] is a flow-based traffic analyzer that performs traffic mining and statistical analysis enabling troubleshooting and anomaly detection for large-scale networks. *Big-DAMA* [5] is a framework for scalable online and offline data mining and machine learning supposed to monitor and characterize extremely large network traffic datasets.

Automata-based Analysis. Timed automaton is one of the natural representations of the behavior models for communication protocols. For example, [14] uses timed automata to model parallel systems and to detect errors by verifying the satisfaction of given properties. However, they do not assume to learn the model automatically. Another work [21] proposes a heuristic state-merging algorithm that learns the model automatically. They are using NetFlow records and time windows to create models that are later used to detect malware and infected hosts. [18] uses a model described by timed automata to diagnose errors. The system monitors several sensors which values are converted into timed sequences to be accepted by the timed automata, which are able to detect violations of the measured values to the predefined model.

3 Model representation

Diagnosed protocols are described using models that define the protocols' communications as pair sequences. Each pair consists of a request and a reply message, as shown in Figure 1. These requests and replies are pre-specified message types specific for each protocol. In addition to the original paper, models will take the form of a timed finite automaton, which, in addition to the message order, will also contain timestamp - time since the last reply was received. The finite automaton will process the input sequence and will traverse through the model states. The result of the traverse process will be the result of diagnostics.

Each model processes a message sequence that is distinguished by a 6-tuple: source and destination IP address, source and destination port, L4 protocol, and session ID. The session ID is an optional parameter specified for each protocol to distinguish multiple conversations that are transmitted within a single connection. When transferring multiple conversations over a single connection, the model does not describe the entire connection, but only individual conversations.

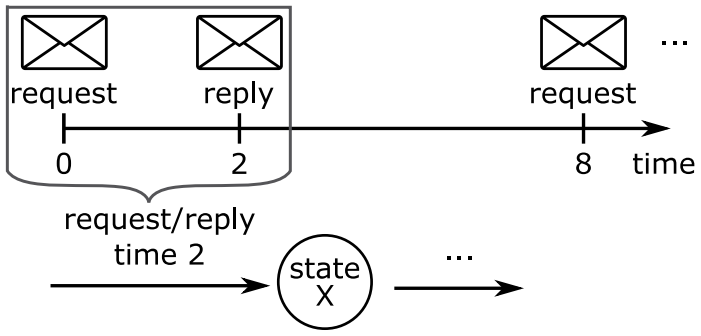

Fig. 1. An illustration of a protocol messages conversion into a finite state automaton. Requests and replies are paired together with the time of their arrival since the last pair.

The finite automaton works with the input alphabet, which is a pair of request and reply values. Both the request and the reply values are composed of packet fields, such as the value of the "*ftp.request.command*" attribute for the FTP request and value of the "*ftp.response.code*" attribute for its reply. If the input symbol (request/reply pair) is repeated (which might mean periodic reports), the model will contain a transition to the same state.

For each request/reply pair, the time since the last reply message or the beginning of the communication is calculated. Using this time, the interval at which the message must arrive for the finite automaton to transition through the state is calculated. Before calculating the interval range, it is necessary to calculate a minimum, a maximum, and a square root of standard deviation from the time values. The interval is calculated in the range from "$minimum - \sqrt{std\ deviation}$" to "$maximum + \sqrt{std\ deviation}$", including extreme values. If the number of values is less than 5, the interval is from zero to infinity (all pairs will match this interval).

A model has a form of a timed finite automaton [22, Def.6.4] — a 6-tuple $(S, S_0, \Sigma, \Lambda, C, \delta)$, where:

- S is a finite set of states,
- $S_0 \in S$ is an initial state,
- Σ is a finite input alphabet ($\Sigma \cap S = \emptyset$, $\epsilon \notin \Sigma$), where Σ = (request, reply) and ϵ is an empty value,
- Λ is a finite output alphabet ($\Lambda \cap S = \emptyset$, $\epsilon \notin \Lambda$), where Λ = error description and ϵ is an empty value,
- C is a finite state of clocks,
- $\delta: S \times (\Sigma \cup \epsilon) \times \Phi(C) \rightarrow S \times \Lambda^* \times 2^C$ is a transition function mapping a triplet of a state, an input symbol (or empty string), and a clock constraint over C to a triplet of a new state, an output sequence, and a set of clocks to be reset. It means, given a specific input symbol, δ shifts the timed transducer from one state to another while it produces an output if and only if the specified clock constraint hold.

4 System Architecture

This section describes the architecture of the proposed system which learns from communication examples and diagnoses unknown communications. In this extended version, the architecture now works with timed information inside automata's transitions, and a new concept of model generalization is described. The system takes PCAP files as input data, where one PCAP file contains only one complete protocol communication. An administrator marks PCAP files as correct or faulty communication examples before model training. The administrator marks faulty PCAP files with error description and a hint on how to fix the problem. The system output is a model describing the protocol behavior and providing an interface for using this model for the diagnostic process. The diagnostic process takes a PCAP file with unknown communication and checks whether this communication contains an error and if yes, returns a list of possible errors and fixes.

The architecture, shown in Figure 2, consists of multiple components, each implementing a stage in the processing pipeline. The processing is staged as follows:

- **Input data processing** - Preprocessing is responsible for converting PCAP files into a format suitable for the next stages. Within this stage, the input packets are decoded using protocol parser. Next, the filter is applied to select only relevant packets. Finally, the packets are grouped to pair request to their corresponding responses.
- **Model training** - The training processes several PCAP files and creates a model characterizing the behavior of the analyzed protocol. The output of this phase is a protocol model.
- **Diagnostics** - In the diagnostic component, an unknown communication is analyzed and compared to available protocol models. The result is a report listing detected errors and possible hints on how to correct them.

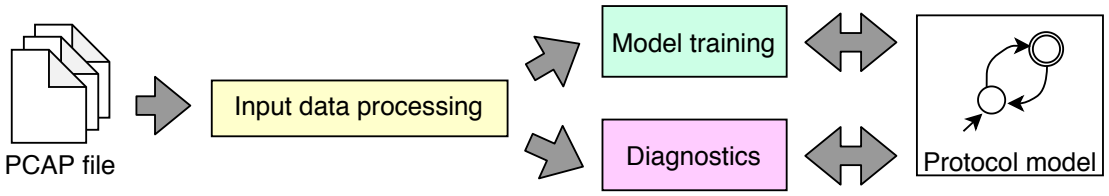

Fig. 2. After the system processes the input PCAP files (the first yellow stage), it uses the data to create the protocol behavior model (the second green stage) or to diagnose an unknown protocol communication using the created protocol model (the-third purple stage). [13]

In the rest of the section, the individual components are described in detail. Illustrative examples are provided for the sake of better understanding.

4.1 Input Data Processing

This stage works directly with PCAP files provided by the administrator. Each file is parsed by *TShark* [4] which exports decoded packets to JSON format. The system further processes the JSON data by filtering irrelevant records and pairs request packets with their replies. The output of this stage is a list of tuples representing atomic transactions.

We have improved the data pairing process in this extended paper to support timed transitions in the model. The system calculates time between the arrival time of the current and the last reply message. For the first reply message within the communication, the time since the beginning of the communication is used. In the case requests do not have corresponding replies, the system uses the requests arrival times. The result of the pairing process is a sequence of pairs with time information, where each pair consists of one request and one reply. The Figure 3 shows an example of this pairing process.

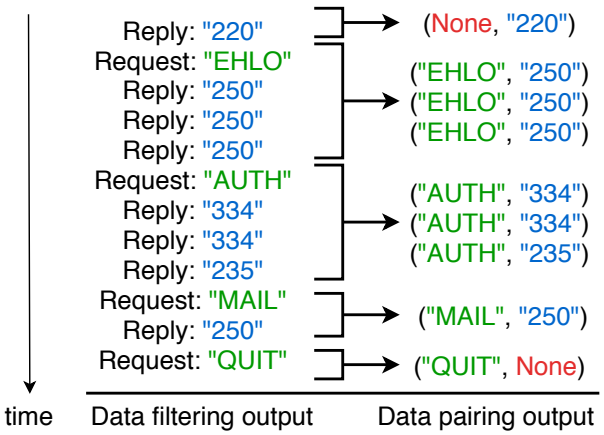

Fig. 3. An SMTP communication in which the client authenticates, sends an email and quits. The left part of the example shows a list of requests and replies together with the time of their arrival in the protocol-independent format. The right part shows a sequence of paired queries with replies, which are the output of the *Input Data Porcessing* stage. For each pair, time since the last pair is also saved. The system pairs one request and one reply with the special *None* value.

4.2 Model Training

After the *Input Data Processing* stage transformed input PCAP files into a list of request-response pairs, the *Model Training* phase creates a protocol model. For example, we can consider regular communication traces that represent typical POP3 protocol operations with the server: the client is checking a mail-box, downloading a message or deleting a message. The model is first created for regular communication and later extended with error behavior.

[4] https://www.wireshark.org/docs/man-pages/tshark.html

Learning from traces with expected behavior. The model creation process begins by learning the protocol behavior from input data representing regular communication. The result of this training phase is a description of the protocol that represents a subset of correct behavior. The model is created from a collection of individual communication traces. When a new trace is to be added, the tool identifies the longest prefix of the trace that is accepted by the current model. The remaining of the trace is then used to enrich the model.

During a traverse within the model, the time attribute of each request-reply pair is added to transitions (each transition has an auxiliary variable containing a list of time attributes). If the number of saved time values within a transition is greater than 5, the time interval of the model transition is recalculated as described in Section 3.

Model generalization. Unfortunately, *TShark* marks some unpredictable data (e.g., authentication data) in some protocols as regular requests and does not clearly distinguish between them. These values are a problem in later processing because these unpredictable values create ungeneralizable states during the model learning phase. Therefore, all transitions that contain requests with unpredictable values are removed from the model and replaced by new transitions.

An unpredictable request value is a request value which is contained inside only one transition - no matter the previous state, the next state, and the reply value. The wildcard value will replace these request values. The time interval of the transition is kept at value from zero to infinity. Which requests contain unpredictable values is determined during the learning process of the model. During this process, the amount of times a request value is being used (no matter the current automata state) is counted (count 1 = unpredictable).

Multiple transitions with unpredictable requests and an identical reply value may originate from a single finite automata state. In this case, all these transitions with the next finite automata states are merged. The merging idea is displayed in Figure 4. After all input traces are used for the model to learn, there is a state from which four transitions are originating. The gray dashed lines are transitions that occurred only once within the input. These two transitions contain various request values, but the same reply value ("OK"). After generalizing these three transitions, a new transition containing the request wildcard value and the "OK" reply value will be added to the model.

When traversing through an automaton, in each state, transitions with explicit commands are checked as first. When no match is found, the model checks if there is a wildcard command value and a reply value for the current state.

Learning the errors. After the system learns the protocol from regular communication, the model can be extended with error traces. The system expects that the administrator prepares that error trace as the result of previous (manual) troubleshooting activities. The administrator should also provide error description and information about how to fix the error.

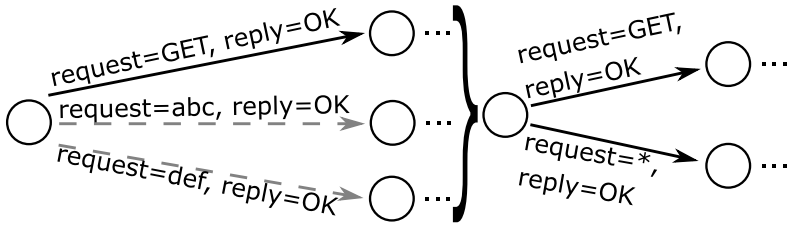

Fig. 4. Illustration of replacing unpredictable requests by a wildcard value (*). The replaced transitions are merged into one generic transition.

When extending the model with error traces, the procedure is similar to when processing correct traces. Automaton attempts to consume as long prefix of input trace as possible ending in state s. The following cases are possible:

- *Remaining input trace is not empty*: The system creates a new state s' and links it with from state s. It marks the new state as an "error" state and labels it with a provided error description.
- *Remaining input trace is empty*:
  - State s is error state: The system adds the new error description to existing labeling of an existing state s.
  - State s is correct state: The system marks the state as *possible error* and adds the error description.

4.3 Diagnostics

After the system creates a behavioral model that is extended by error states, it is possible to use the model to diagnose unknown communication tracks. The system runs diagnostics by processing a PCAP file in the same way as in the learning process and checks the request-reply sequence with their time attributes against the automaton. Diagnostics distinguishes between these classes:

- **Normal:** The automaton accepts the input trace and ends in the correct state.
- **Error:** The automaton accepts the input trace and ends in the error state.
- **Possible error:** The automaton accepts the input trace and ends in the possible error state. In this case, the system cannot distinguish if the communication is correct or not. Therefore, the system reports an error description from the state and leaves the final decision on the user.
- **Unknown:** The automaton does not accept entire the input trace, which may indicate that the trace represents a behavior not fully recognized by the underlying automaton.

It is important to notice that during the traverse within the automaton, the time attribute of each request-reply pair is compared with time constraints. In case the time attribute does not fulfill the constraint, the model generates a warning message. However, the diagnostic process does not stop, and the traverse process continues to the next state in the same way as if the time constraint was fulfilled.

5 ALGORITHMS

This section provides algorithms for (i) creating a model from normal traces, (ii) generalization of the model, (iii) updating the model from error traces and (iv) evaluating a trace if it contains an error. The algorithms are based on algorithms from the original paper. The difference is that in this version, they need to work with timed transitions. All presented algorithms work with a model that uses a deterministic timed finite automaton (DTFA) as its representation.

To simplify algorithms' codes, we have defined time interval $\langle 0; \infty \rangle$ as the default interval. If the interval is not specified, the model uses this value which has less priority when traversing throw the model states. Only when there is no match with a specific interval, the system checks default values.

5.1 Adding Correct Traces

Algorithm 1 takes the input model (DTFA) and adds missing transitions and states based on the input sequence (P). The algorithm starts with the *init_state* and saves it into the *previous_state* variable. The *previous_state* variable is used to create a transition from one state to the next. In each loop of the while section, the algorithm assigns the next pair into the *current_state* variable until there is no next pair. From the *previous_state* and the *current_state*, the *transition* variable is created, and the system checks if the DTFA contains this *transition*. If the DTFA does not contain it, it is added together with the *time_value*. Otherwise, the new *time_value* is added to the *transition*. If at least five time values are saved, the *time_interval* is calculated and applied to the *transition*.

Before continuing with the next loop, the *current_state* variable is assigned to the *previous_state* variable. The updated model will be used as the input for the next input sequence. After processing all the input sequences, which represent normal behavior, the resulting automaton is a model of normal behavior.

Algorithm 1 Updating model from the correct traces

Inputs: P = query-reply pairs sequence with time value; DTFA = set of the transitions
Output: DTFA = set of the transitions
previous_state = *init_state*
while *not at end of input P* **do**
  current_state = get next pair from *P*
  transition = *previous_state* → *current_state*
  if *DTFA does not contain transition* **then**
    add *transition* to DTFA and save *time_value* to the *transition*
  else
    add *time_value* to the saved times in *transition*
    if *saved times >= 5* **then**
      calculate the *time_interval* constraint and apply it to the *transition*
  previous_state = *current_state*
end
return DTFA

Algorithm 2 Generalization of the model

```
Inputs: DTFA = set of transitions
Output: DTFA = set of transitions
foreach transition ∈ DTFA do
    if transition contains only one time then
        new_transition = make copy of transition
        remove transition from DTFA
        replace request in new_transition by wildcard
        if DTFA does not contain new_transition then
            add new_transition to DTFA
end
return DTFA
```

5.2 Model Generalization

The Algorithm 2 takes all transitions from a model one by one (variable *transition*) calculates the number of times each *transition* was used, and checks whether the *transition* was used only once (contains only one time value). Only one time value means that in all of the input traces, the *transition* was used only once. The model creates a new copy of the *transition* (variable *new_transition*) and removes the old one.

The wildcard value replaces the request value in the *new_transition*. The algorithm checks whether the model contains this *new_transition*, and if not, it is inserted into the model. This presence control ensures that a single transition replaces multiple ungeneralizable states with a wildcard request value.

Algorithm 3 Extending the model with error traces

```
Inputs: P = query-reply pairs sequence; DTFA = set of transitions; Error = description
        of the error
Output: DTFA = set of transitions
previous_state = init_state
while not at end of input P do
    current_state = get next pair from P
    transition = previous_state → current_state
    if DTFA contains transition then
        if transition fulfills time_interval then
            if transition contains error then
                append error to transition in DTFA
                return DTFA
            previous_state = current_state
        else
            add transition to DTFA and mark it with error
            return DTFA
    else
        add transition to DTFA and mark it with error
        return DTFA
end
return DTFA
```

5.3 Adding Error Traces

The Algorithm 3 has one more input (*Error*), which is a text string describing a user-defined error. The start of the algorithm is the same as in the previous case. The difference is in testing whether the automaton contains the *transition* specified in the input sequence. If so, the system checks whether the *transition* fulfills the *time_interval*. This time interval checking is an improvement of the algorithm from the previous paper. Only when the *time_interval* is fulfilled, the system checks to see if the saved *transition* also contains errors. In this case, the algorithm updates the error list by adding a new *error*. Otherwise, the algorithm continues to process the input string to find a suitable place to indicate the error. If the *transition* does not fulfill the *time_interval* restriction or the *transition* does not exist, it is created and marked with the specified *error*.

5.4 Testing Unknown Trace

The Algorithm 4 uses previously created automaton (DTFA variable) to check the input sequence P. According to the input sequence, the algorithm traverses the automaton and checks whether the transitions contain errors. If an error in some transition is found, the system returns an errors description messages (*errors*) to the user. If the *transition* was not found, the algorithm returns an unknown error. In this case, it is up to the user to analyze the situation and possibly extend the automaton for this input.

In this extended paper, the system also verifies if the input sequence fulfills transitions time restrictions. With each *transition*, the time value is compared to the *time_interval*. If the *transition* does not fulfill the *time_interval*, the system creates a warning message to the user. More than one warning message can be generated because the generating of warning messages does not stop the diagnostic process.

Algorithm 4 Checking an unknown trace

Inputs: P = query-reply pairs sequence; DTFA = set of transitions
Output: Errors = one or more error descriptions
previous_state = *init_state*
while *not at end of input P* **do**
  current_state = get next pair from *P*
  transition = *previous_state* → *current_state*
  if *DTFA contains transition* **then**
    if *transition doesn't fulfill time_interval* **then**
      create warning that *transition* does not matched the interval and continue
    if *transition contains error* **then**
      return *errors* from *transition*
    previous_state = *current_state*
  else
    return "unknown error"
end
return "no error detected"

6 EVALUATION

We have implemented a proof-of-concept tool which implements the Algorithm 1, 2, 3, and 4. In this section, we provide the evaluation of our proof-of-concept tool to demonstrate that the proposed solution is suitable for diagnosing application protocols. Another goal of the evaluation is to show how the created model changes by adding new input data to the model. We have chosen four application protocols with different behavioral patterns for evaluation.

The results from the original's subsections 5.1-5.3 are the same and still valid. From this reason the subsections 6.1 and 6.3 are the same as in the original paper, and in the subsection 6.2 the figure showing the model's complexity during the model training is omitted. The new content is in the following subsections. Section 6.4 tests the benefit of using finite automata as the model by detecting a performance problem inside a communication. The last section 6.5 tries to verify whether the proposed approach is somehow usable for encrypted traffic.

6.1 Reference Set Preparation and Model Creation

Our algorithms create the automata states and transitions based on the sequence of pairs. The implication is that repeating the same input sequence does not modify the learned behavior model. Therefore, it is not important to provide a huge amount of input files (traces) but to provide unique traces (sequences of query-reply pairs). We created our reference datasets by capturing data from the network, removing unrelated communications, and calculating the hash value for each trace to avoid duplicate patterns. Instead of a correlation between the amount of protocols in the network and the amount of saved traces, the amount of files correlates with the complexity of the analyzed protocol. For example, hundreds of DNS query-reply traces captured from the network can be represented by the same query-reply sequence $(A\ type\ query, No\ error)$.

After capturing the communication, all the traces were manually checked and divided into two groups: (i) traces representing normal behavior and (ii) traces containing some error. In case the trace contains an error, we also identified the error and added the corresponding description to the trace. We split both groups of traces into the training set and the testing set.

It is important to notice that the tool uses traces to create a model for one specific network configuration and not for all possible configurations. Focus on a single configuration results in a smaller set of unique traces and smaller created models. This allows an administrator to detect situations which may be correct for some network, but not for a diagnosed network, e.g., missing authentication.

6.2 Model Creation

We have chosen the following four request-reply application protocols with different complexity for evaluation:

– **DNS**: Simple stateless protocol with communication pattern - domain name query (type A, AAAA, MX, ...) and reply (no error, no such name, ...).

- **SMTP**: Simple state protocol in which the client has to authenticate, specify email sender and recipients, and transfer the email message. The protocol has a large predefined set of reply codes resulting in many possible states in DTFA created by Algorithm 1 and 2.
- **POP**: In comparison with SMTP, the protocol is more complicated because it allows clients to do more actions with email messages (e.g., download, delete). However, the POP protocol replies only with two possible replies (+OK, -ERR), which reduce the number of possible states.
- **FTP**: Stateful protocol allowing the client to do multiple actions with files and directories on server. The protocol defines many reply codes.

Table 1. For each protocol, the amount of total and training traces is shown. These traces are separated into *proper* (without error) and *failed* (with error) groups. The training traces are used to create two models, the first without errors and the second with errors. The *states* and *transitions* columns show the complexity of the models. [13]

Protocol	**Total traces**		**Training traces**		**Model without error states**		**Model with error states**	
	Proper	**Failed**	**Proper**	**Failed**	**States**	**Transitions**	**States**	**Transitions**
DNS	16	8	10	6	18	28	21	34
SMTP	8	4	6	3	11	18	14	21
POP	24	9	18	7	16	44	19	49
FTP	106	20	88	14	33	126	39	137

The proof-of-concept tool took input data of selected application protocols and created models of the behavior without errors and a model with errors. The Table 1 shows the distribution of the input data into a group of correct training traces and a group of traces with errors. Remaining traces will be later used for testing the model. The right part of the table shows the complexity of the generated models in the format of states and transitions count.

Based on the statistics of models, we have made the following conclusions:

- transitions sum depicts the model's complexity better than the state's sum;
- there is no direct correlation between the complexity of the protocol and the complexity of the model. As can be seen with protocols DNS and SMTP, even though the model SMTP is more complicated than DNS model, there were about 50% fewer unique traces resulting in a model with 21 transitions, while the DNS model consists of 34 transitions. The reason is that one DNS connection can contain more than one query-reply and because the protocol is stateless, any query-reply can follow the previous query-reply value.

Part of the original paper is a figure with four charts outlining the same four protocols, as displayed in Table 1. These four charts show the progress of increasing the model size and decreasing the number of diagnostic errors

when new traces are added to the model. The model creation process was split into two parts: training from traces without errors and learning the errors.

6.3 Evaluation of Test Traces

Table 2 shows the amount of successful and failed testing traces; the right part of Table 2 shows testing results for these data. All tests check whether:

1. a successful trace is marked as correct (TN);
2. a failed trace is detected as an error trace with correct error description (TP);
3. a failed trace is marked as correct (FN);
4. a successful trace is detected as an error or failed trace is detected as an error but with an incorrect error description (FP);
5. true/false (T/F) ratios which are calculated as $(TN+TP)/(FN+FP)$. T/F ratios represents how many traces the model diagnosed correctly.

Table 2. The created models have been tested by using testing traces, which are split into *proper* (without error) and *failed* (with error) groups. The correct results are shown in the true negative (TN) and true positive (TP) columns. The columns false positive (FP) and false negative (FN) on the other side contain the number of wrong test results. The ratio of correct results is calculated as a true/false ratio (T-F *ratio*) . This ratio represents how many testing traces were diagnosed correctly. [13]

Protocol	**Testing traces**		**Testing against model without error states**					**Testing against model with error states**				
	Proper	**Failed**	**TN**	**TP**	**FN**	**FP**	**T-F ratio**	**TN**	**TP**	**FN**	**FP**	**T-F ratio**
DNS	6	2	4	2	0	2	75 %	4	1	1	2	63 %
SMTP	2	1	2	1	0	0	100 %	2	1	0	0	100 %
POP	6	2	6	2	0	0	100 %	6	2	0	0	100 %
FTP	18	6	18	6	0	0	100 %	18	5	1	0	96 %

TN - true negative, TP - true positive, FN - false negative,
FP - false positive, T-F ratio - true/false ratio

As the columns T-F ratio in Table 2 shows, most of the testing data was diagnosed correctly. We have analyzed the incorrect results and made the following conclusions:

– **DNS**: False positive - One application has made a connection with the DNS server and keeps the connection up for a long time. Over time several queries were transferred. Even though the model contains these queries, the order in which they came is new to the model. The model returned an error result even when the communication ended correctly. An incomplete model causes this misbehavior. To correctly diagnose all query combinations, the model has to be created from more unique training traces.

- **DNS**: False positive - The model received a new SOA update query. Even if the communication did not contain the error by itself, it is an indication of a possible anomaly in the network. Therefore, we consider this as the expected behavior.
- **DNS**: False negative - The situation was the same as with the first DNS False positive mistake - the order of packets was unexpected. Unexpected order resulted in an unknown error instead of an already learned error.
- **FTP**: False negative - The client sent a PASS command before the USER command. This resulted in an unexpected order of commands, and the model detected an unknown error. We are not sure how this situation has happened, but because it is nonstandard behavior, we are interpreting this as an anomaly. Hence, the proof-of-concept tool provided the expected outcome.

All the incorrect results are related to the incomplete model. In the stateless protocols (like DNS), it is necessary to capture traces with all combinations of query-reply states. For example, if the protocol defines 10 types of queries, 3 types of replies, the total amount of possible transitions is $(10 * 3)^2 = 900$. Another challenge is a protocol which defines many error reply codes. To create a complete model, all error codes in all possible states need to be learned from the traces.

We have created the tested tool as a prototype in Python language. Our goal was not to test the performance, but to get at least an idea of how usable our solution is, we gathered basic time statistics. The processing time of converting one PCAP file (one trace) into a sequence of query-replies and adding it to the model took on average 0.4s. This time had only small deviations because most of the time took initialization of the *TShark*. The total amount of time required to learn a model depends on the amount of PCAPs. At average, to create a model from 100 PCAPs, 30 seconds was required.

6.4 Timed transitions

For this test, we took a model of the SMTP protocol from the previous test, and we have extended it with new PCAP files. These new PCAP files contain two problems that could not be detected without a timed finite automata model:

1. **overloaded SMTP server** - all requests from the server have a high delay;
2. **overloaded authentication LDAP server** - the SMTP server responds to requests at an average speed, but user authentication, which uses an external LDAP server takes considerably longer.

Unfortunately, we do not have PCAP files with these errors from a real production network, so we had to create them. We achieved this by manually overloading the SMTP server, LDAP server, or creating a delay for the communication between these two servers.

The part of the extended model that covers authentication problems is displayed in Figure 5. Red colored transitions cover situations where, regardless

of the authentication type and authentication result, a slow response is detected. The model describes two authentication methods: simple authentication (name and password in one message) and login authentication (name and password sent separately). As described in Algorithm 3, these new transitions have a time interval with the value $< 0; \infty >$. Therefore all traces that do not match the original time restrictions are matched by these new transitions.

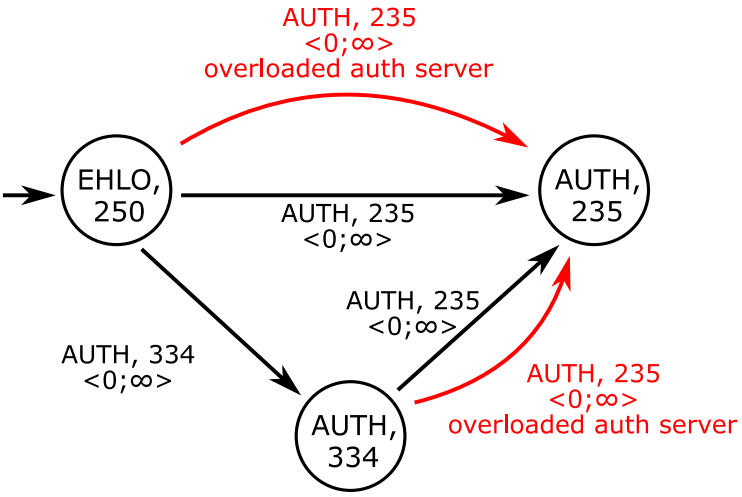

Fig. 5. The segment of the SMTP model which contains new transitions and states related to the high delay from the authentication server.

We have tested the created model on other captured PCAP files. With overloaded LDAP servers or high communication latency between an SMTP and an LDAP server, the model correctly detected a problem related to the authentication. When the SMTP server was overloaded, the model correctly detected overload at the beginning of the communication.

However, the extended SMTP model was not able to correctly diagnose a situation where the beginning of the communication was OK, and the overload of the SMTP server began during client authentication. Although other delayed responses followed the delayed response for authentication, the system stopped at the first error and erroneously detected an authentication problem.

6.5 Encrypted data diagnostics

We have performed another type of evaluation aimed at verifying if the method proposed by us applies to encrypted traffic or not. As described in the previous sections, the diagnostic process uses request-reply values, but we are not able to detect this in encrypted communication. To overcome this limitation, we have proposed a modification to the model in the way that the model uses the size of the encrypted data (TLS record size) instead of the request-reply value.

Because we only consider the size of the application data, which can easily vary even if the request value or the reply value is the same, it is neces-

sary to work with a range of values. We are using an algorithm similar to the one used to calculate the range for time intervals. From the set of values, we calculate the minimum, maximum, and square root of the standard deviation. The range of the interval that will accept messages will have a value ranging from "$minimum - \sqrt{std\ deviation}$" to "$maximum + \sqrt{std\ deviation}$". The difference from the calculation of the time intervals is that with a range of application data sizes, the interval is calculated from even a single value, and it is not required to have at least five values.

With this modified approach of diagnostics, we are not able to diagnose such a range of errors as in unencrypted traffic. However, we are still able to obtain at least basic information about the state of communication. We have based this idea on the fact that protocol communication between endpoints goes through different states. Protocol standards specify these states and their order. Diagnostics of encrypted communication, do not analyze exactly what caused the error but only when (or in which state) the error occurred.

As in the case of unencrypted communications, a model should be created only from traces belonging to one service (on a single server) and applied to the same service. With other configurations, the content of messages can be different, which would cause different sizes of the messages themselves.

To verify the idea of diagnostics based on the size of application data, we have captured ten correct and three error SMTP communication traces. From these communications, a model was created, which is shown in Figure 6. The Figure shows three detectable errors that also separate the SMTP protocol states - welcoming client, user authentication, and e-mail sending. Based on this test, we have reached the conclusion that the approach is usable. However, to use the model in the real-world, the model should be trained from more traces.

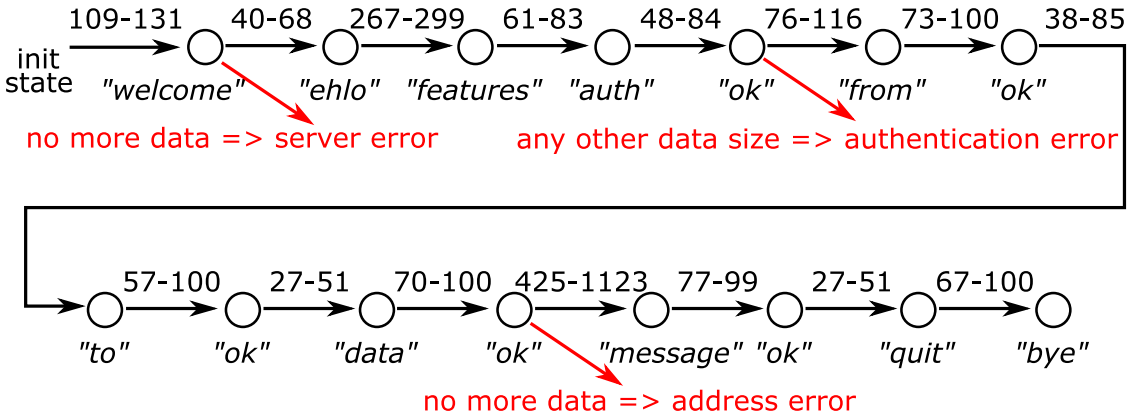

Fig. 6. The segment of the SMTP model which contains new transitions and states related to the high delay from the authentication server.

7 DISCUSSION

This chapter describes some of the topics we have come across when developing and using the tool.

7.1 Fully trained model

One of the fundamental questions when using the tool is when the model is fully (or for X%) trained and when it is possible to switch from training mode to diagnostic mode. The simplest way of specifying how much percent the model is trained is by calculating all possible transitions. Transitions are connecting any two states, which are defined by request/reply values. The total number of states is $requests\_count * replies\_count$, and the total number of transitions is $states\_count^2$. Of course, many combinations of requests and replies do not make sense, but the algorithm can never be sure which combinations are valid and which are not. The problem with counting all possible combinations is that without predefined knowledge of the diagnosed protocol, the tool can never be sure if all possible requests and replies have already been seen or not.

One way to determine whether a model is trained without knowing the total number of states is by checking the list of trained states when processing new input data. If the system has not detected a new state for a certain amount of iterations, it will declare that the list is complete, and the model is fully trained. Here comes the problem of determining how long to wait for a new value.

Basically, there are three approaches that can be combined:

A1 amount of new files - waiting for X new files to be processed (e.g., 100);
A2 training duration- waiting for an X lasting interval;
A3 unique amount of clients - waiting for X unique clients (e.g., 10).

Unfortunately, each of these approaches has drawbacks that cannot be eliminated entirely:

eA1 If most (or all) new files have the same content type (for example, the same client queries the same DNS translation type), then the number of these files is not important. We have partially solved this problem by creating communications fingerprints and ignoring duplicate fingerprint files. This is why the evaluation section describes so few unique communications. As an example, we take the SMTP protocol. Most conversations had an identical pattern - welcoming the client, user authentication, and sending an email. The responses to all commands were without any error. Although the welcome message (timestamp), login information, email addresses, and email content varied from one communication to the next, the fingerprint was the same as all of this data was deleted in the *Input Data Processing* stage. So even though we had dozens of these conversations, we counted them as just one conversation.

eA2 By taking communications carried during a limited time frame, e.g. 24 hours, we may not cover situations that arise less frequently or irregularly. For example, SMTP clients can process requests to send a message even when the client is offline and then send these messages at once when the client is online again. However, the situation, when a client sends multiple messages at once, does not occur often, and a 24-hour window might not be enough.

eA3 This criterion can only be applied if a large number of clients connect to a single server. In the case of a pre-defined server-server communication, this criteria makes no sense. Even with a larger amount of clients, all clients may use the same application, the same settings, and perform the same activity.

In our opinion, the best option is to combine all three mentioned approaches and select parameters so that the data sample used is relevant and that the model is trained within an acceptable time. For example, waiting for 100 unique sequences in the SMTP protocol or 1000 communications if only one communication happens per day is meaningless. However, even in this case, we are not able to capture the following situations:

- Protocol updates can introduce new version of the protocol which may introduce new types of commands or responses.
- Another version (e.g., by an update) of the application or a brand new application will appear on the network. This may cause the client to start to communicate with the server with a different pattern of behavior.
- Some types of errors are associated with less frequently used features, which occur very irregularly. Such errors are hard to catch and get into the model.

From our experience, it is not possible to determine when the model is fully trained or at least trained from X%. Even if the model does not grow for a long time, it can suddenly expand by processing a new trace (new extensions, programs with specific behavior, program updates).

Nevertheless, the model incorporates means to train even in the diagnostic phase (when the tool is deployed). An administrator that encounters a false error can always improve the model. Consequently, as time passes, the model can adapt to handle infrequent communications and protocol/application updates.

7.2 Data labeling

During model training, an administrator needs to determine if there is an error in conversation manually. If an error is detected, an administrator creates a description of this error. This process is time-consuming and requires knowledge of the modeled protocol and computer networks in general. However, it is important to realize that in the case of manual diagnosis, the administrator has to perform a similar diagnosis. Hence, our approach does not introduce additional requirements for administrators' skills. Therefore, we do not think that the need to manually mark communications is a disadvantage of our method.

Another possible way to label data can be by applying artificial intelligence or machine learning. However, we think that even with machine learning, supervised learning has to be used. Therefore, it is still necessary to analyze the content of the communications manually and instruct the algorithm. Another question is, how easy it is for the network administrator to work with artificial intelligence, and whether network administrators without programming knowledge understand working with machine learning.

7.3 Model of models

As part of our proposed approach, we do not model relationships between individual communications or between different protocols. Each model describes one particular communication with one protocol. However, there are more complex errors that cannot be detected or diagnosed by analyzing just a single communication. An example is downloading a web page content. This activity can consist of multiple individual communications: user authentication, HTML page download, and download of other elements such as images or scripts. Another example is that during communication with an application server, the server establishes another connection to the RADIUS server to authenticate the user.

To be able to diagnose problems that are spread across multiple communications correctly (even over multiple protocols), it is necessary to create a model which will consist of several models describing individual communications ("model of models"). This high-level model can check one communication and, based on its result, launch another model for the following communication or generate a diagnostic report.

7.4 Another usage of models

The proposed models do not apply to network diagnosis only. Another application is the security analysis. The model can be trained to accept only communications which fulfill the security policy. Other communications that are not accepted by the trained model are reported as possibly dangerous. Another type of security analysis is by visualizing the model and employing a manual analysis. Our tool can export models to a format suitable for graphical visualization. From the trained models, an administrator can make some deductions. For example, if some users are not using the recommended authentication or some communications contain outdated commands.

Another possible model usage is related to time transitions within the model. We think it makes sense to investigate whether it is possible to use models for profiling communications. For example, in the case of FTP communication, if the browsing and downloading of files are without delays caused by user interaction, it is possible to associate such communication with a tool that automatically browses and downloads server content.

8 CONCLUSIONS

In the presented paper, we have proposed an automatic method for generating automata from network communication traces and their use in the network diagnostic process. The diagnostic system is designed to learn from both normal error-free communication sequences as well as from erroneous traces in order to create an automata-based model for the communication protocol behavior. The states in the automaton can be labeled with additional information that provides diagnostic information for the error detected.

The method requires network traces prepared by an expert to create a good model. The expert is expected to annotate network traces and label the known errors. The current model is only applicable to query-response protocols and those that provides a sufficient amount of information to observe their state. We demonstrated that if the model is created based on the reasonable sample of good and error behavior it can be used in any network environment.

We have implemented the method in a proof-of-concept tool[5] and use it in a set of experiments for demonstration purposes. The tool has been tested on a limited set of application protocols of different types, e.g., e-mail transfer, file download, domain name resolution. Experiments show that the suitability and usability of the model heavily depend on the network protocol. Although the model typically does not cover all possible scenarios, it is useful for diagnosis of repetitive error. As the model can learn errors during deployment, an administrator does not have to deal with errors not encountered during learning phase more than once.

ACKNOWLEDGEMENTS

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4In-novations excellence in science - LQ1602".

References

1. Aggarwal, B., Bhagwan, R., Das, T., Eswaran, S., Padmanabhan, V.N., Voelker, G.M.: NetPrints: Diagnosing home network misconfigurations using shared knowledge. Proceedings of the 6th USENIX symposium on Networked systems design and implementation **Di**(July), 349–364 (2009), http://portal.acm.org/citation.cfm?id=1559001
2. Anand, A., Akella, A.: Net-replay: a new network primitive. ACM SIGMETRICS Performance Evaluation Review (2010). https://doi.org/10.1145/1710115.1710119
3. Antunes, J., Neves, N., Verissimo, P.: Reverx: Reverse engineering of protocols. Tech. Rep. 2011-01, Department of Informatics, School of Sciences, University of Lisbon (2011), http://hdl.handle.net/10451/14078

[5] https://github.com/marhoSVK/semiauto-diagnostics

4. Burschka, S., Dupasquier, B.: Tranalyzer: Versatile high performance network traffic analyser. In: 2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016 (2017). https://doi.org/10.1109/SSCI.2016.7849909
5. Casas, P., Zseby, T., Mellia, M.: Big-DAMA: Big Data Analytics for Network Traffic Monitoring and Analysis. Proceedings of the 2016 Workshop on Fostering Latin-American Research in Data Communication Networks (ACM LANCOMM'16) (2016). https://doi.org/2940116.2940117
6. Chen, M., Zheng, A., Lloyd, J., Jordan, M., Brewer, E.: Failure diagnosis using decision trees. International Conference on Autonomic Computing, 2004. Proceedings. pp. 36–43 (2004). https://doi.org/10.1109/ICAC.2004.1301345, http://ieeexplore.ieee.org/document/1301345/
7. Comparetti, P.M., Wondracek, G., Krügel, C., Kirda, E.: Prospex: Protocol specification extraction. 2009 30th IEEE Symposium on Security and Privacy pp. 110–125 (2009)
8. Cui, W., Kannan, J., Wang, H.J.: Discoverer: Automatic Protocol Reverse Engineering from Network Traces. USENIX Security (2007). https://doi.org/"Protocol-Independent Adaptive Replay of Application Dialog"
9. De Paola, A., Fiduccia, S., Gaglio, S., Gatani, L., Lo Re, G., Pizzitola, A., Ortolani, M., Storniolo, P., Urso, A.: Rule based reasoning for network management. In: Seventh International Workshop on Computer Architecture for Machine Perception (CAMP'05). pp. 25–30 (July 2005). https://doi.org/10.1109/CAMP.2005.47
10. Dhamdhere, A., Teixeira, R., Dovrolis, C., Diot, C.: NetDiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data. Proceedings of the 2007 ACM CoNEXT (2007). https://doi.org/10.1145/1364654.1364677
11. Dong, C., Dulay, N.: Argumentation-based fault diagnosis for home networks. In: Proceedings of the 2nd ACM SIGCOMM Workshop on Home Networks. p. 37–42. HomeNets '11, Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/2018567.2018576, https://doi.org/10.1145/2018567.2018576
12. El Sheikh, A.Y.: Evaluation of the capabilities of wireshark as network intrusion system. Journal of Global Research in Computer Science **9**(8), 01–08 (2018)
13. Holkovič, M., Ryšavý, O., Polčák, L.: Using network traces to generate models for automatic network application protocols diagnostics. In: Proceedings of the 16th International Joint Conference on e-Business and Telecommunications Volume 1: DCNET, ICE-B, OPTICS, SIGMAP and WINSYS. pp. 43–53. SciTePress - Science and Technology Publications (2019), https://www.fit.vut.cz/research/publication/12012
14. Ivković, N., Milić, L., Konecki, M.: A timed automata model for systems with gateway-connected controller area networks. In: 2018 IEEE 3rd International Conference on Communication and Information Systems (ICCIS). pp. 97–101. IEEE (2018)
15. Kim, S., Ahn, S.j., Chung, J., Hwang, I., Kim, S., No, M., Sin, S.: A rule based approach to network fault and security diagnosis with agent collaboration. In: Kim, T.G. (ed.) Artificial Intelligence and Simulation. pp. 597–606. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
16. Krueger, T., Gascon, H., Krämer, N., Rieck, K.: Learning stateful models for network honeypots. In: Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence. p. 37–48. AISec '12, Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2381896.2381904, https://doi.org/10.1145/2381896.2381904

17. Lodi, G., Buttyon, L., Holczer, T.: Message Format and Field Semantics Inference for Binary Protocols Using Recorded Network Traffic. In: 2018 26th International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2018 (2018). https://doi.org/10.23919/SOFTCOM.2018.8555813
18. Lunze, J., Supavatanakul, P.: Diagnosis of discrete–event system described by timed automata. IFAC Proceedings Volumes **35**(1), 77–82 (2002)
19. Ming Luo, Danhong Zhang, G.P.L.C.: An interactive rule based event management system for effective equipment troubleshooting. Proceedings of the IEEE Conference on Decision and Control **8**(3), 2329–2334 (2011). https://doi.org/10.1007/s10489-005-4605-0
20. Orzach, Y.: Network Analysis Using Wireshark Cookbook. Packt Publishing Ltd (2013)
21. Pellegrino, G., Lin, Q., Hammerschmidt, C., Verwer, S.: Learning behavioral fingerprints from netflows using timed automata. In: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM). pp. 308–316. IEEE (2017)
22. Polčák, L.: Lawful Interception: Identity Detection. Ph.d. thesis, Brno University of Technology, Faculty of Information Technology (2017), https://www.fit.vut.cz/study/phd-thesis/679/
23. Procházka, M., Macko, D., Jelemenská, K.: IP Networks Diagnostic Communication Generator. In: Emerging eLearning Technologies and Applications (ICETA). pp. 1–6 (2017)
24. Samhat, A., Skehill, R., Altman, Z.: Automated troubleshooting in WLAN networks. In: 2007 16th IST Mobile and Wireless Communications Summit (2007). https://doi.org/10.1109/ISTMWC.2007.4299084
25. łgorzata Steinder, M., Sethi, A.S.: A survey of fault localization techniques in computer networks. Science of computer programming **53**(2), 165–194 (2004)
26. Tong, V., Tran, H.A., Souihi, S., Mellouk, A.: Network troubleshooting: Survey, Taxonomy and Challenges. 2018 International Conference on Smart Communications in Network Technologies, SaCoNeT 2018 pp. 165–170 (2018). https://doi.org/10.1109/SaCoNeT.2018.8585610
27. Traverso, S., Tego, E., Kowallik, E., Raffaglio, S., Fregosi, A., Mellia, M., Matera, F.: Exploiting hybrid measurements for network troubleshooting. In: 2014 16th International Telecommunications Network Strategy and Planning Symposium, Networks 2014 (2014). https://doi.org/10.1109/NETWKS.2014.6959212
28. Wang, Y., Zhang, Z., Yao, D.D., Qu, B., Guo, L.: Inferring protocol state machine from network traces: A probabilistic approach. In: Lopez, J., Tsudik, G. (eds.) Applied Cryptography and Network Security. pp. 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
29. Wondracek, G., Comparetti, P.M., Kruegel, C., Kirda, E.: Automatic network protocol analysis. In: Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08 (2008)
30. Xiao, M.M., Yu, S.Z., Wang, Y.: Automatic network protocol automaton extraction. In: NSS 2009 - Network and System Security (2009). https://doi.org/10.1109/NSS.2009.71
31. Zeng, H., Kazemian, P., Varghese, G., McKeown, N.: A survey on network troubleshooting. Technical Report Stanford/TR12-HPNG-061012, Stanford University, Tech. Rep. (2012)

A.7 Network Problem Diagnostics using Typographic Error Correction

Authors: Holkovič Martin, Ing. (70%), Bohuš Michal, Ing. (20%), Ryšavý Ondřej, doc. Ing., Ph.D. (10%)
Abstract: Detecting and correcting network and service availability issues is an essential part of the network administrator's daily duty. One of the causes of errors can be the user herself providing incorrect input. The present work describes a new diagnostic method that detects incorrectly inserted inputs observed in network-related data, e.g., network traffic, log files. The proposed method aims to detect incorrect words in domains, login names, or email addresses. First, we describe how to detect possible incorrect words. For each such detected word, a list of correct candidates is created based on edit distance. Next, the correction method selects the best word by scoring candidates based on the probability of occurrence in the given context. The proposed method was implemented as a prototype and tested on words created using real user activities. The evaluation demonstrates that this approach can substantially reduce the time needed to identify this kind of errors.
Keywords: Computer network errors, network diagnostics, typographic error correction, end-user data diagnostics.
Published in: 17th International Conference on Network and Service Management (CNSM 2021), Izmir, Turkey
Conference rating: B(Core), B4(Qualis)
ISSN: 2165-963X
ISBN: 978-3-903176-36-2

Network Problem Diagnostics using Typographic Error Correction

Martin Holkovič
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
iholkovic@fit.vutbr.cz

Michal Bohuš
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic

Ondřej Ryšavý
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
rysavy@vutbr.cz

***Abstract*—Detecting and correcting network and service availability issues is an essential part of the network administrator's daily duty. One of the causes of errors can be the user herself providing incorrect input. The present work describes a new diagnostic method that detects incorrectly inserted inputs observed in network-related data, e.g., network traffic, log files. The proposed method aims to detect incorrect words in domains, login names, or email addresses. First, we describe how to detect possible incorrect words. For each such detected word, a list of correct candidates is created based on edit distance. Next, the correction method selects the best word by scoring candidates based on the probability of occurrence in the given context. The proposed method was implemented as a prototype and tested on words created using real user activities. The evaluation demonstrates that this approach can substantially reduce the time needed to identify this kind of errors.**

***Index Terms*—computer network errors, network diagnostics, typographic error correction, end-user data diagnostics**

I. Introduction

Errors in computer networks can be caused by a lot of different types of faults. System or device failure can prevent networks from working, some services may become unavailable, and user experience could be negatively affected. Due to the great variety of errors, there is no single procedure or path to detect the cause of all network-related errors.

Some of the errors can be caused by incorrect user input. For example, users can often type a wrong URL into a web browser, wrong credentials into a mail transfer agent, or an incorrect phone ID for a VoIP call. When a user incorrectly specifies some of this data, the requested service will be unavailable or not work as expected. Because some of the information entered by a user is necessary to initiate a network connection or is transmitted in network messages, it is possible to detect wrong information by analyzing of suitable data sources, e.g., network packets, NetFlow records, or log files.

We propose a new diagnostic method that applies typographical error detection techniques and correction commonly used in spell-checkers. The proposed method can inform the network administrator about the incorrectly inserted values from end-users by detecting typographical errors in user data. At the same time, it provides a suggestion for a possible correction. This can be useful for the administrator as he can automatically find out whether the error is created by a typographical error or if there is another reason.

The paper contribution is as follows. We developed a method for detecting errors in user data. The method is similar to that used by spell-checker systems. For each possible incorrect word, the algorithm creates a list of possible correct word candidates selected using the number of edit operations as a distance between the words. Among these candidates, the correct word is selected using the ranking method. The problem with current solutions is that they do not work well without context (individual words) and with words that are not based on grammar rules (e.g., email address xholko00@fit.vutbr.cz).

The method's direct application is to validate typographical errors in domain names and usernames, which enables us to apply it to a wide range of applications and services. Regarding domain names, except for the purely diagnostic use case, the method can be employed for detecting various types of malicious activities, such as an IDN homograph attack, typosquatting, and other forms of domain phishing attacks.

This paper presents the principles and design of a new diagnostic tool that focuses on detecting typing errors. The method relies only on passive data sources (captured traffic, NetFlow records, log files). Similarly, as in [8], our method will be able to work in a learning mode for which it will require error-free data. The proposed method is considered as a complementary tool for the existing systems.

The paper is organized as follows: The next section describes related work mentioning the key results in computer network diagnostics. Sec. III provides background information about research done on spell checking. Sec. IV defines the principles behind the developed tool. Sec. V presents the architecture of the tool. In Sec. VI, the approach and results of the evaluation are provided. Finally, Sec. VII concludes the paper by discussing the contribution and identifying further improvements.

II. Related Work

Because computer networks are complex systems, errors are unavoidable and sooner or later occur [1]. Errors can affect network performance or user experience, which can cause other network problems [2]. Therefore, it is necessary to find and correct errors correctly. This is addressed in diagnostics, which is an important part of network management [1].

Fault diagnosis is a time-consuming activity that requires in-depth knowledge of network operation. Administrators often do not have the appropriate tools or knowledge to diagnose network problems, and they would like to have sophisticated automated tools to help them diagnose those errors [3]. In case of insufficient automatic tools, problems must be diagnosed manually, for example, using the Wireshark tool [4], [5].

Network problems can be divided into application and network problems [6]. An example of application problems is a broken server service or a badly configured client software. Problems related to network infrastructure fall under network problems. An error can be caused by a human - unintentional (misconfiguration) / intentional (attack), or a device failure [7].

Currently, a single tool that can diagnose all kinds of errors does not exist. Researchers have so far developed a wide range of diagnostic tools [8]. The tools can be divided either according to what data they work with - network packets, SDN data, NetFlow records, log files [9]–[12], or how they access the data - passive, active, and hybrid [6], [13], [14].

III. Background

Researchers are working on correcting typographical errors since the 1960s [15]. Correcting this type of error is based on the fact that people make mistakes when typing input data. Because users are often unaware of the error, a wrong value can cause a problem. One of the best-known examples of typographic error correction in misspelled words is in text editors such as Microsoft Office Word [16]. There are also other uses, such as Optical Character Recognition (OCR) [17] or typographical error password tolerance [18], [19].

Kukich [15] has divided the problem of typing errors correction into three categories:

1) nonword error detection (wrong word detection only),
2) isolated-word error correction (finding the wrong word and proposing a correction),
3) context-dependent correction (finding the wrong word and offering a correction based on the text's context).

Word correction extends error detection by offering correction for the word with a typographical error. It tries to guess what word the user could have thought of, solves candidates' selection, and possibly chooses the best candidate. In addition to common language words, searching for typographical errors also makes sense in other data types such as numeric values [21] or domain names [22].

Correcting typographical errors consists of three parts: 1) finding an error; 2) creating a list of candidates; 3) rating of individual candidates [15].

A. Finding an error

To detect a wrong word, it is necessary to model the words of the language. The model needs to detect several sources of error, such as pronunciation similarity, typographic similarity, or user's bad habits [23]. The most commonly used models are based on vocabularies or n-grams [15].

1) N-grams: N-grams are n character long substrings of words. The most commonly used are bigrams (n=2) and trigrams (n=3). Finding errors based on n-grams works by generating all n-grams from the analyzed word and comparing them with a previously created model. If the n-gram is not present in the model or only with a small occurrence, it is expected that this is an error. An example is a "zfq" trigram, which is not common in English. To create an n-gram model, a huge text containing only words without errors is needed.

2) Dictionaries: In this case, the model is the dictionary itself, created by storing all the unique words from the error-free text. If a word cannot be found inside a dictionary during the check phase, it is considered a typing error [24], [25].

A search response time may be a problem when using the dictionary as a model [18]. A common technique to speed up dictionary searches is to split one large dictionary into several smaller ones. One way to divide is by word length [31]. For example, when searching for a word of 5 characters in length, it will only search within words of the same length.

The basic and often used technique to access a dictionary is by using a hash table. A dictionary can also be implemented by tree structures such as a ternary search tree [32], trie, binary search trees sorted by frequency, trees with words or characters in nodes, bloom filters [33], or finite state machines [34].

3) Types of errors: The two basic types of errors that automatic word correction focuses on are [32], [35]:

- nonword error - if an error occurs, a non-existent word is created (e.g., hello - henlo);
- real-word error - an error will result in an existing word (e.g., pay - day).

Errors that result in an existing word are more challenging to detect because it is necessary to know the text's context. Mitton analyzed the errors from tests filled out by students at the age of 15 and found that the real-world error accounted for 40% of the errors [36]. The types of errors vary depending on the environment in which they occurred; for example, if it is a writer, most of the errors will be due to pressing the wrong key. On the other hand, OCR errors will be based on similar-looking characters such as B and 8. Word-bounded errors are a specific kind of error where the space between words is missing, or space is shifted [32].

There are three kinds of mistakes that result in a typographical error [15]:

- typographical errors are those where the user types the word "henlo" instead of "hello", where it is assumed that the author knows how the word is spelled and only pressed a bad character on the keyboard;
- cognitive errors arise from insufficient knowledge of the user - the user did not know the correct form of the word;
- phonetic errors occur when replacing a character with another similar-sounding one.

B. Creating a list of candidates

When a misspelled word is detected, it is determined from which possible words the misspelled word could have been formed. For this reason, functions that determine the distance

between two words are being used. The earliest and one of the best-known ways to determine the distance of two strings is the Levenshtein distance [37]. The principle is that we specify the minimum number of operations required to transform one string into another. Allowed operations are insertion, deletion, and substitution. Wagner and Fisher [38] also called the distance the editing distance. There are other types of editing distances, such as Hamming [39], [40], Dameraou-Levenstein [41], or Spring-Winkler [42].

The test with data from Birkbeck Spelling Error Corpus [43] showed that out of 1000 errors in the English language, 742 are one-character, 201 two-character, 44 three-character, 9 four-character, 4 five-character [24]. In the case of TshwaneDJe Sesotho sa Leboa corpus [44], from the 908 errors in domain names were 804 one-character, 78 two-character, and 26 three-character. Further analysis of large texts' errors confirmed that 80-95% of errors are single-character errors [27]. The results show that it is most important to check words with a short editing distance when searching for errors.

C. Rating of individual candidates

After generating all words from which it is possible to create a searched word with a typographical error within a certain editing distance, it is necessary to rate them. The purpose of the rating is to find the most likely word from which the error originated. Research is focused not only on the accuracy of results but also on the speed of the ranking process [45], [46].

If the user interactively selects one of several options, it is possible to adapt to a specific user and his type of errors [47]. Another learning option introduces a learning mode during which all the found words are saved as correct [25]. In addition to classic methods, it is also possible to use machine learning and artificial intelligence [48], [49].

Even though it is impossible to use the language's grammar to find the best candidates for correction [31], [50], it is still possible to use the probabilities of specific errors to prefer some words before others. The most basic heuristic is to adjust the weights of the editing distance according to the type of operation. For example, several sources suggest that each operation has a different probability from others. Specifically, their probability is in the following order (from most to least probable): substitution, deletion, insertion, and transposition [36], [44], [51]–[53].

IV. Methodology

This work aims to create a new method for searching for typographical errors and proposing their correction in data that contain settings and parameters of various network applications. It does not make sense to analyze all data that are coming from the user. For example, hashed passwords, timestamps, or any random values. In contrast to detecting typographical errors in plain text, the method must work with words for which it is impossible to apply grammar from plain text, such as domain names or IP addresses. Simultaneously, all words will be processed individually, so it will not be possible to use the context in which the words are located. Since the typing error is not created by a machine but by a human, it is necessary to focus on the user's data. We have recognized the following categories of data: IP address, transport port, domain name, username, generic number, generic string.

The term word in this work means a character string belonging to one of the described categories. It will also be possible to create combined categories from these categories. An example of a combined category is an email address consisting of a login name, a "@" delimiter, and a domain name. With combined categories, the values are broken down into basic categories and stored in the appropriate dictionaries. This approach allows knowledge sharing, so if someone on the network visits the "company.local" domain name, the system can find a typographical error in the "user@copmany.local" email, even if the email has never been seen before.

A. Detecting an error

Our proposed method works on the principle of dictionaries. What is in the dictionaries is valid and correct; what is not is considered a typing error. N-gram analysis could also be used for this purpose, but for words that are not part of the natural language (domain name, IP address), the N-grams would not work correctly. Many correct words would be marked as incorrect, and at the same time, many incorrectly marked as correct. Dictionaries are also used to filter out possible corrections for detected incorrect words, and a suitable replacement is eventually selected using heuristic methods.

Words are not all stored together but are stored by word category in several smaller dictionaries. This means that only a dictionary containing email addresses is used to determine if an email is a correct word or a typing error. Each dictionary further consists of two parts - predefined and learned.

The predefined part of a dictionary contains words that are expected in a given context. In the case of words of a common language, this is a list of all grammatically correct words. This can be a list of the most visited domain names or transport ports of well-known services for other data types. These dictionaries are expected to be customized by the administrator based on the data that are present inside the network. For example, the administrator can enter all company's usernames.

The dictionaries' learned parts are not created manually by administrators, but their content is created by learning from the input data. One way to teach a tool the correct data is to use only data for learning that is not reported as problematic. The downside is that end-users do not report all problems because they can fix some typographical errors themselves. The second way of learning is that the administrator manually verifies which data is correct and which is not. However, this is unrealistic due to the huge amount of network data in real networks. The last option is to use an external tool that evaluates each communication, whether it has finished successfully or with an error [55]. In this case, the tool would automatically learn only from correct communications.

B. Generating candidates

For all words detected as words with an error, finding suitable candidates for correction in the dictionaries is neces-

sary. Candidates are all possible words from which the typing error could be created with a certain editing distance. We are working only with words that have an edit distance of less than or equal to 2. Simultaneously, to consider a word a correct candidate, the word needs to be in the appropriate dictionary.

The creation of candidates is based on applying reverse operations to basic operations (insert, delete, replace, and swapping of two adjacent characters). For example, if we want to determine if the word "ello" was formed by omitting the first letter, the method tries a reverse operation (in this case, the opposite of omitting is the insertion of a character). After trying all characters, the method detects that inserting the letter "h" before the word "ello" creates the valid word "hello". To try all the possibilities, it is necessary to try all operations on all word positions and to use all combinations of characters. Candidates with edit distance 1 are created by one operation and candidates with distance 2 by two operations.

Our method uses two different ways of creating candidates. The first method applies all operations to the detected word with an error and tries to obtain words present in the dictionaries [3]. The second method works oppositely and takes correct words from dictionaries, and calculates the number of operations required to create a correct word from the misspelled word. The second method is more complex, but with fewer words in the dictionaries, this method is significantly faster. The amount of words for which the second method is faster than the first one is described in the evaluation section.

C. Choosing the best candidate

After creating candidates for word correction, it is necessary to determine the best replacement from them. The easiest way would be to select the candidate with the shortest editing distance. The problem is that several candidates may have the same minimum distance. Based on this information alone, we cannot select the most suitable candidate.

We propose a scoring algorithm that tries to assign higher scores to the candidates with higher probability. At first, the algorithm assigns a basic score for each candidate. In the next step, the algorithm takes the operations that are needed to change the word with an error to the correct word (e.g., insert letter "h" at the beginning of the word and replace the third letter "i" with the letter "l"). For each of the operations, a penalty is deducted from the score. The main idea is that each operation has a different penalty based on probability.

We have analyzed several research papers and identified which errors are more probable than others. The list of these errors is in Table I. If the operation contains any of these errors, its penalty is reduced. Each operation may be attributed to several probable errors, and the penalty is reduced for each of them. For example, an error in the username "matousek", where the user mistakenly writes "mat0usek" is represented as an operation to replace the fourth character "o" with "0". This operation contains up to three errors with a higher probability: 1) error is not in the first character; 2) characters "o" and "0" are in a fat finger distance; 3) characters "o" and "0" are visually very similar. All errors have the same weight, and the same amount reduces the penalty for each of them.

TABLE I
LIST OF ERRORS WITH A HIGHER PROBABILITY

Error type	Example
Duplication [21]	44 > 444
Shift [21]	441 > 411
Pluralism [15]	money > moneys
Doubling or skipping a voice [36]	scissors > sissors
Fat-finger [56], [57]	help > hewlp
Transcription error [19]	1 > l
Phonetic error [32]	blake > brake
Keyboard layout [19], [32]	zoomba > yoomba
Capitalization [19]	home > HOME
Error not in first character [31]	config > comfig

V. TOOL ARCHITECTURE

This section describes the architecture of the designed tool, which implements the proposed diagnostic method. The tool is intended to help network administrators with finding and fixing a problem on the network. A typical use case for this is when a service does not work for an end-user. The end-user reports the problem to the administrator, who uses data analysis to determine that the service is not working for the user because he entered some data incorrectly due to a typing error. Finally, incorrect words with possible corrections will be displayed to the administrator. Before using the tool, the administrator needs to know what data to analyze and in which location this data can be found.

The proposed tool consists of several independent blocks, shown with their interconnection in Fig. 1. In the left part of the figure, there is the input data, which, after processing, continues to other blocks according to the selected mode. The result of the first (learning) mode is a set of learned words, and the result of the second (diagnostic) mode is a list of detected words with a typographical error (or errors) and the proposed corrections.

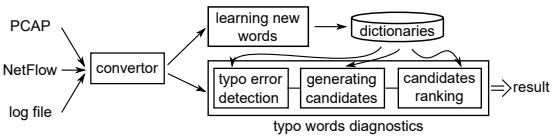

Fig. 1. An architecture of the proposed tool which consists of multiple blocks.

A. Convertor

The first part of the tool is used to convert data from different sources that use a different format into a common format. The common format will allow the rest of the tool to work with data no matter its source. For this purpose, a converter is used. In addition to data converting, it filters the input data because it does not make sense to analyze all the data from the sources (e.g., timestamps or password hashes). The format is based on JSON, consisting of keys (categories)

and lists of values belonging to those categories. The rest of this subsection describes currently supported sources.

1) Network packets: The analysis will look at user data entered into the application and transported in network packets. An example is the configuration of an email client, where the user enters the login name. Packets can be saved in the PCAP format for later analysis. We are using TShark to convert packets saved in PCAP format into JSON format, which is better for data analysis. We have chosen TShark because we do not need to implement our custom protocol parsers.

2) NetFlow records: Another way to analyze network data is to use NetFlow technology, which aggregates packets according to common properties and reduces the amount of analyzed data. The tool is working with NetFlow records that are saved in the CSV format. This CSV format consists of columns representing NetFlow attributes and rows that contain individual records. It is important to note that current state-of-the-art implementations of NetFlow monitoring are not sampling the analyzed data even for high-speed networks and are exporting application fields, e.g., the domain name from DNS or email address from SMTP.

3) Log files: The advantage of processing log files over network data is that packet analysis cannot process any data in the case of using an encrypted connection. However, the data is written to the log files in a readable form. Processing log files allows the analysis to look for errors regardless of the network topology or protocols used. Although log files are in text form by default, their format is not uniform. Each application creates logs differently, and therefore it is up to the administrator to specify exactly how the information should be searched for and extracted from the data. For this purpose, we used regular expressions to specify which part of the record will be analyzed.

B. Learning mode

After the converter processes the data, it is possible to start one of the two modes. The first mode is responsible for learning new words and saving them into dictionaries. Words are stored in the appropriate dictionary according to their category and length. For example, when saving the domain name ”milk.com”, the word’s length is determined (it is 8) and saved to the dictionary containing only domain names of the same length. The goal of dividing the dictionaries by word length is to make it easier to find words with similar lengths.

C. Diagnostic mode

After the tool has learned the correct words, it is possible to proceed to the diagnostics mode. The second possibility is that the learning mode was not used at all, and the administrator manually predefined all the correct words. This consists of three parts - typing error detection, generating candidates, and candidate ranking. Each of these parts is using dictionaries that contain both predefined and learned words.

1) Typing error detection: The first step in diagnostics is to find out which words are correct and which words have a typographical error. To do this, it is necessary to load dictionaries of correct words. When the program is loading dictionaries, the individual words are loaded into Python sets. The search for whether a word is in the dictionary is performed as a search for an element in a set. The benefit of implementing a search over a set is that it is possible to search for multiple elements in a very short amount of time. In our test, we searched for 1000 elements in less than 1 ms, while the data structure with 100.000 words took up less than 16 MB of operational memory.

2) Generating candidates: The second step in diagnosis mode is to obtain potential candidates to correct the detected typographical error. The proposed method is using two algorithms for generating candidates. The first one is applying operations to the word with a typographical error to create a correct word. The second algorithm takes words from dictionaries and calculates the number of operations (edit distance) to create a word with error. The reason why we are using the second algorithm is the expectation that it is faster than the first one at the moment when there are not many words in the dictionaries. Therefore, we performed a test in which we try to find out the limit when the second algorithm is faster and, conversely, when the second algorithm starts to be slower than the first algorithm.

The result of the test is shown in Fig. 2. The first algorithm’s time depends only on the word’s length and is shown by a line with circle points. The other lines show the second algorithm’s time according to the number of words with the same length. It can be seen that the time complexity of the first algorithm is approximately the same as the complexity of the second algorithm when there are 25000 words with a similar length as the analyzed word in the dictionary.

Based on the test, we chose a 25000 word limit to determine which algorithm to use. Because the method searches for candidates up to edit distance 2, it is possible to count only words that are not shorter or longer by two characters than the detected word with an error. When processing a word with a typographical error of length N, the tool determines whether the number of words in the dictionary of length N-2 to N+2 is less than or greater than 25000.

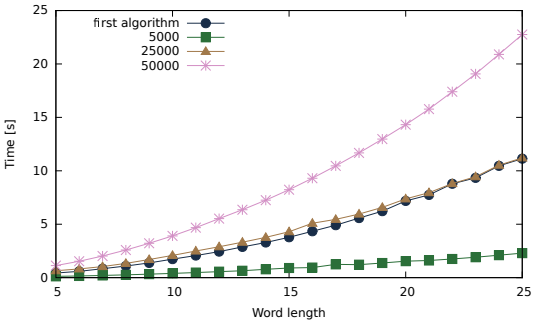

Fig. 2. Comparison of the generation time of candidates of the first and second algorithm.

3) Candidate ranking: The algorithm for candidate ranking works in two stages: assigning a base score and adjusting it based on the error type. The first stage is simple. If a word was found in the learned part of the dictionary, a score of 25 is used. Otherwise, a score of 20 is used for words from the predefined part. Afterward, it is checked whether the word is of the type domain name, string, or username and whether the candidate has the same phonetic sound as the typographical error (according to the Metaphone algorithm). If the phonetic sound is the same, the value 3 is added to the score.

In the second stage, the method takes the operations needed to change the word with an error to the correct word and calculates the penalty that will be reduced from the score. The penalty has a default value of 10. A list of conditions defining more probable errors is checked, and for each fulfilled condition, the penalty is reduced by 1. The system checks the following conditions that define more probable errors:

- inserted character is the same as the adjacent character;
- inserted character is in the fat finger distance as the adjacent character;
- changed character is in the fat finger distance as the original character;
- changed character is on the same position in different keyboard layouts (qwerty-qwertz);
- mistake is not on the first character;
- error is around the separator (dot, hyphen or underscore);
- inserted character is same as the both adjacent characters;
- inserted character is in prefix or suffix of the word;
- replaced character is visually similar to the original character;
- removed character is same as the adjacent character;
- both the original and replaced character are different, but both are the same as one of the adjacent character.

Before substituting the penalty from the score, the penalty is multiplied by the coefficient based on the operation type: insert = 0.6, replace = 0.3, delete = 0.5, transpose = 0.7).

There is one exception for which the previous calculation is not used. When a word with typographical error is capitalized (caps-lock was enabled on the user's keyboard). This operation is not a basic operation used in the edit-distance calculation, and therefore calculating scores based on these operations does not make sense. In this case, the algorithm assigns the maximum score of 25 to all capitalized words. This approach will make these words always the best candidates.

D. Example

The outputs from the individual tool parts can be seen in Fig. 3, which also shows the whole method's main idea. First, words from the converter belonging to the category domain name are displayed, which are searched for in dictionaries. The domain name "amzon.com" is not in the dictionaries and is considered a typographical error. All possible words with a maximum editing distance of 2 that can be created from the domain name are generated. All these words are searched for in dictionaries, and only six are considered as correction candidates. Candidates are ranked, and the one with the best score (amazon.com) is selected as a replacement for the original misspelled word.

Fig. 3. Output's example of individual parts of the architecture.

VI. Evaluation

This section shows that the presented diagnostic method can detect typing errors caused by end-users. The first test shows that the created heuristic for evaluating candidates sorts the individual errors as expected. The second test tests the success rate of correcting typographical errors on the created dataset. In the third test, the speed of the whole tool is measured. The last test tried to correct typographical errors in IP addresses and port numbers.

A. Ranking of candidates

The first test aims to verify the correctness of the created heuristic for scoring correction candidates. We have manually applied several errors to the word "google.com", and a score heuristic algorithm was run over the words with errors. The aim is not to verify the exact values of the calculated score but to determine whether the order of the candidates according to the calculated score adequately reflects the probability of the type of typographical error that occurred. The higher the score, the higher the probability of the candidate should be. For example, one inserted character from a fat finger distance should have a better rating (higher score) than deleting two characters.

The first test result is shown in Table II, containing incorrect words, a description of operations, and a calculated score. The calculated score sorts the items in the table. There is no exact way to determine whether the created order is correct or not. However, according to intuition, it can be stated that more probable errors compared to the words with the lowest rating are placed in the first rows.

B. Candidates proposing accuracy

The second test aims to verify that the created heuristic selects the correct candidates as a replacement for the detected errors. We have created a script that applied typing errors from regular English text to the list of domain names for this test. As a list of domain names, we have used the most visited 797641 domain names from Alexa [1]. A dataset with 2455 typographical errors from 1922 words from English Wikipedia from Roger Mitton was used to simulate real errors. Each typing error in this dataset also contains the correct form of the word.

[1]http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

TABLE II
SCORING VARIOUS CANDIDATES FOR THE GOOGLE.COM DOMAIN NAME

Score	Wrong word	Operation
25.0	GOOGLE.COM	Capitalization
23.9	g0ogle.com	Substitution for a similar symbol
23.6	ggogle.com	Substitution with a duplicated symbol
23.6	giogle.com	Substitution in a fat finger distance
23.3	gwogle.com	Substitution
22.0	gogle.com	Deletion of the repeated symbol
21.8	gooogle.com	Insertion of a repeated symbol
21.5	goole.com	Deletion
21.2	giigle.com	Substitution in a fat finger distance
21.2	google.comp	Insertion at the end of the word
20.6	gooqgle.com	Insertion
19.7	ogogle.com	Transposition
18.5	gopople.com	Insertion in a fat finger distance
17.0	ggle.com	Deletion
15.2	goqoqgle.com	Insertion

The script for generating domain names with typing errors consists of four steps:

1) The list containing pairs with correct and error words has been created. E.g., the correct word ”guard” was paired with the misspelled word ”gaurd”. Only words with a maximum editing distance of 2 have been used.
2) Domain names were analyzed to determine whether they contain any correct words from the created list. E.g., the domain name *theguardian.com* contains the word ”guard”. The list of pairs was always searched in random order to prevent repeating the same type of errors.
3) If the domain name contained any valid word, this word was replaced with a misspelled word. E.g., the domain name *theguardian.com* was changed to *thegaurdian.com*.
4) After 1000 different domain names have been created with typing errors. The process has stopped.

Out of 1000 created domain names, nine domain names were such that the new word was also a valid domain name even after applying the typing error. It means that the new domain name was in the same list of domain names from Alexa, and therefore it is not possible to mark those domain names as incorrect. Of the remaining 991 words, 967 cases were correctly corrected. Four misspelled words have had more than one candidate with the best score, and one of those candidates was the correct word. The remaining 20 domain names were incorrectly repaired. For example, the ”udemy.com” domain name had a typographical error ”udemi.com”, which was corrected to ”udemu.com” because replacing the letter ”i” with the letter ”u” is more likely because it is located near to it (based on the QWERTY keyboard layout). The results are shown in Table IV, and several correctly repaired domain names with different types of errors are shown in Table III.

Another test to verify the best candidates' accuracy was made on a list of 371 usernames of employees and Ph.D. students within our faculty. All usernames are based on real names. This list was used as a dictionary to check whether the checked username is correct or not. We have randomly selected 100 usernames and applied the typing errors in the same way as in the test with domain names. From these 100 usernames with errors, 100 were correctly detected as incorrect and 99 were correctly corrected. The results are also shown in Table IV. The one incorrectly repaired username was ”ikuceran” which was created from the username ”ikucera”, however our tool repaired it as ”ikuceraj”.

TABLE III
SELECTED DOMAINS WITH ERRORS THAT WERE CORRECTLY FIXED

Wrong word	Heuristic rules	Correct word
voice-real.com	transposition; similar phonetic	voice-reel.com
panbda.tv	insert; fat finger distance	panda.tv
chasr.com	substitution; fat finger distance	chase.com
ilvoepdf.com	transposition	ilovepdf.com
weatherr.com	insert; letter duplication	weather.com
gole.co.kr	2x delete; removed repeating symbol	google.co.kr

TABLE IV
RESULT OF TESTING THE CORRECTNESS OF PROPOSED CORRECTIONS FOR DETECTED TYPOGRAPHICAL ERRORS IN DOMAIN NAMES AND USERNAMES

Data type	domains	usernames
Word count	1000	100
Detected as error	991	100
Detected as correct word	9	0
Correct best candidate	967	99
Multiple best candidates	4	0
Wrongly repaired	20	1
Success rate	97.5% (967 of 991)	99% (99 of 100)

We have detected up to 97.5% of errors in domain names and 99% of username errors in misspelled words generated from sources that contained real user typographical errors. However, it is unnecessary to achieve 100% accuracy to use the tool because even with a lower success rate, the tool can make it easier to diagnose errors. We do not consider typographical errors words that are also correct as errors. For example, if a user types ”google.cz” instead of ”google.ca”, it is impossible to detect an error because both are valid domain names. Therefore we did not count them when calculating the success rate.

C. Usage on other data types

The third test aimed to verify functionality on the domain name of IP addresses and port numbers. It turned out that the proposed method of error detection does not apply to real networks very well. There were two main reasons. The first problem was to obtain relevant data to which the tool could be applied. Most networks use DNS protocol and default port numbers, so the end-users rarely use IP addresses and ports.

The second problem was that even though some data has been collected, there is only a small difference between the individual values. Usual domain names, login names, and emails are very different and easily distinguishable. For example, login names may look like ”vecerav”, ”polcak”, or ”iletavay”. On the other hand, IP addresses inside a

single network are usually assigned from the same subnet, e.g., ”147.229.176.14”, ”147.229.176.19,” or ”147.229.176.8”. With these values, there is a high probability that the value with an error will also be a valid value. Even if the error will be detected, it is almost impossible to say the correct value without additional knowledge.

VII. Conclusions

This work aimed to create a method for detecting typing errors in computer network applications appearing in various data sources. The tool is intended to complement existing systems for network monitoring and troubleshooting. The tool focuses on data entered directly by end-users and then transmitted over a network or stored somewhere in application servers. An example is the configuration of the email client, which is provided manually by the user. For example, when an end-user incorrectly configures such an application and does not know why the application is not working, the end-user contacts an administrator. The administrator will use this tool which will automatically check whether the application is not working because of the typing error.

The presented method can be considered a spell-checker for network-related data. The created tool focuses only on detecting nonword errors, so if another correct word is created because of a typing error, it is not identified as an error. The tool checks for each word, whether it is present inside a dictionary, and if not, it is considered a typographical error.

At the beginning of the work, we have identified data types in which it makes sense to search for typographical errors. The most interesting data sources are network packets, NetFlow records, and log files regarding availability and amount of data containing possible user' errors. Each type of data has its advantages and disadvantages, and therefore it is impossible to say that only one specific type of data should be used.

Many kinds of typographical errors can be made when typing on the keyboard. The basic mechanism behind the emergence of such errors was identified by Damerau, who defined it in terms of insertion, transposition, replacement, and deletion of a character. Based on these operations, it is possible to measure the editing distance between two words and find similar words to be used as a correction. When there are multiple candidates for correction, it is necessary to determine the best selection. To solve this selection problem, we have developed an algorithm that considers the probabilities of individual operations' occurrence and looks for the most common types of typographical errors in candidates. Based on the highest evaluation, the best candidate for replacement is selected. In the case of equally rated candidates, there are multiple corrections offered.

The created tool works in two modes. Firstly, the learning mode is applied to create dictionaries of correct words based on the provided input data considered to be correct. Secondly, the diagnostic mode uses a learned model to detect and correct typing errors. The program can also run without the learning phase, in which case it requires a manually created model.

The tool's functionality was tested on errors in domain names and login names created by applying real-user mistakes from the English Wikipedia. Knowing the correct form of each error made it possible to determine whether the tool suggests correct correction candidates. Although the tool's success rate was less than 100% (97.5% and 99%), the tool can still speed up diagnostics by not requiring the administrator to manually analyze the data entered by the end-user.

Although the method can be applied to any data entered by a user, it is not appropriate to look for typographical errors in identifiers from layers other than from the application layer. The reason is that only the application identifiers are optimized for typing by users. There are significant differences between each value, and it is thus possible to easily estimate the intended word in the case of a typing error.

Finally, we have identified the following future works:

- The tool does not allow feedback processing of the previous errors. By marking each correction as right or wrong, the diagnostic method would prioritize previously successful candidates.
- Every word that has been seen during the learning mode is considered correct. However, when learning from data with possible errors, words that were seen only once may indicate a potential typo. It would make sense to penalize less frequent words in dictionaries. However, it would be necessary to deal with words that occur less frequently and be erroneously marked as typos, even if they are already in the dictionary.

Acknowledgment

This work was supported by the BUT FIT grant FIT-S-20-6293, ”Application of AI methods to cyber security and control systems”.

References

[1] Han, Y., Zhao, X. and Li, J., 2018. Computer Network Failure and Solution. Journal of Computer Hardware Engineering, 1(1).

[2] Wang, R., Wu, D., Li, Y., Yu, X., Hui, Z. and Long, K., 2012. Knight's tour-based fast fault localization mechanism in mesh optical communication networks. Photonic Network Communications, 23(2), pp.123-129.

[3] Solé, M., Muntés-Mulero, V., Rana, A.I. and Estrada, G., 2017. Survey on models and techniques for root-cause analysis. arXiv preprint arXiv:1701.08546.

[4] Orzach, Y., 2013. Network Analysis Using Wireshark Cookbook. Packt Publishing Ltd.

[5] Squicciarini, A.C., Petracca, G., Horne, W.G. and Nath, A., 2014, March. Situational awareness through reasoning on network incidents. In Proceedings of the 4th ACM conference on Data and application security and privacy (pp. 111-122).

[6] Van, T.,Tran, H. A.,Souihi, S. A. Mellouk, A. Network troubleshooting:Survey, Taxonomy and Challenges. In: IEEE.2018 International Conference onSmart Communications in Network Technologies (SaCoNeT). 2018, pp. 165–170.

[7] Zeng, H., Kazemian, P., Varghese, G. and McKeown, N., 2012. A survey on network troubleshooting. Technical Report Stanford/TR12-HPNG-061012, Stanford University, Tech. Rep.

[8] Chen, A., Wu, Y., Haeberlen, A., Zhou, W. and Loo, B.T., 2016, August. The good, the bad, and the differences: Better network diagnostics with differential provenance. In Proceedings of the 2016 ACM SIGCOMM Conference (pp. 115-128).

[9] Kögel, J., Including the Network View into Application Response Time Diagnostics using Netflow.

[10] Rudrusamy, G., Ahmad, A., Budiarto, R., Samsudin, A. and Ramadass, S., 2013. Fuzzy based diagnostics system for identifying network traffic flow anomalies. arXiv preprint arXiv:1304.7864.

[11] Qiu, T., Ge, Z., Pei, D., Wang, J. and Xu, J., 2010, November. What happened in my network: mining network events from router syslogs. In Proceedings of the 10th ACM SIGCOMM conference on Internet measurement (pp. 472-484).

[12] Csikor, L. and Pezaros, D.P., 2017, December. End-host driven troubleshooting architecture for software-defined networking. In GLOBECOM 2017 IEEE Global Communications Conference (pp. 1-7). IEEE.

[13] Pullmann, J. and Macko, D., 2018, November. Network tester: A generation and evaluation of diagnostic communication in ip networks. In 2018 16th International Conference on Emerging eLearning Technologies and Applications (ICETA) (pp. 451-456). IEEE.

[14] Traverso, S., Tego, E., Kowallik, E., Raffaglio, S., Fregosi, A., Mellia, M. and Matera, F., 2014, September. Exploiting hybrid measurements for network troubleshooting. In 2014 16th International Telecommunications Network Strategy and Planning Symposium (pp. 1-6). IEEE.

[15] Kukich, K. Techniques for automatically correcting words in text. Acm Computing Surveys (CSUR). ACM. 1992, 24(4), pp.377-439.

[16] Hirst, G. "An evaluation of the contextual spelling checker of Microsoft Office Word 2007." (2008).

[17] Youssef, B. and Alwani, M.. "Ocr post-processing error correction algorithm using google online spelling suggestion." arXiv preprint arXiv:1204.0191 (2012).

[18] Chen, X., Huang, X., Mu, Y. and Wang, D., 2019, August. A Typo-Tolerant Password Authentication Scheme with Targeted Error Correction. In 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE) (pp. 546-553). IEEE.

[19] Chatterjee, R., Athayle, A., Akhawe, D., Juels, A. and Ristenpart, T., 2016, May. pASSWORD tYPOS and how to correct them securely. In 2016 IEEE Symposium on Security and Privacy (pp. 799-818). IEEE.

[20] Ahmad, I., Parvez, M.A. and Iqbal, A., 2019, July. TypoWriter: A Tool to Prevent Typosquatting. In 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC) (Vol. 1, pp. 423-432). IEEE.

[21] Sun, Y.C., Tang, D.D., Zeng, Q. and Greenes, R., 2002. Identification of special patterns of numerical typographic errors increases the likelihood of finding a misplaced patient file. Journal of the American Medical Informatics Association, 9(Supplement_6), pp.S78-S79.

[22] Wang, Y.M., Beck, D., Wang, J., Verbowski, C. and Daniels, B., 2006. Strider Typo-Patrol: Discovery and Analysis of Systematic Typo-Squatting. SRUTI, 6(31-36), pp.2-2.

[23] QasemiZadeh, B., Ilkhani, A. and Ganjeii, A., 2006, June. Adaptive language independent spell checking using intelligent traverse on a tree. In 2006 IEEE Conference on Cybernetics and Intelligent Systems (pp. 1-6). IEEE.

[24] Lianga, H.L., Watsonb, B.W. and Kourieb, D.G., Technical Report Classification for Selected Spell Checkers and Correctors.

[25] Peterson, J.L., 1980. Computer programs for detecting and correcting spelling errors. Communications of the ACM, 23(12), pp.676-687.

[26] Damerau, F.J. and Mays, E., 1989. An examination of undetected typing errors. Information Processing & Management, 25(6), pp.659-664.

[27] Bao, Z., Kimelfeld, B. and Li, Y., 2011, June. A graph approach to spelling correction in domain-centric search. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (pp. 905-914).

[28] Odell, K.M. and Russell, R.C., 1918. Soundex phonetic comparison system. US Patent, 1261167.

[29] Pollock, J.J. and Zamora, A., 1984. Automatic spelling correction in scientific and scholarly text. Communications of the ACM, 27(4), pp.358-368.

[30] Philips, L., 1990. Hanging on the metaphone. Computer Language, 7(12), pp.39-43.

[31] Elmi, M.A. and Evens, M., 1998. Spelling correction using context. In COLING 1998 Volume 1: The 17th International Conference on Computational Linguistics.

[32] Martins, B. and Silva, M.J., 2004, October. Spelling correction for search engine queries. In International Conference on Natural Language Processing (in Spain) (pp. 372-383). Springer, Berlin, Heidelberg.

[33] Mullin, J.K. and Margoliash, D.J., 1990. A tale of three spelling checkers. Software: Practice and Experience, 20(6), pp.625-630.

[34] Aho, A.V. and Corasick, M.J., 1975. Fast pattern matching: an aid to bibliographic search. Communications of ACM, 18(6), pp.333-340.

[35] Jurafsky, D. and Martin, J.H.: Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Pearson Prentice Hall (2009).

[36] Mitton, R. English Spelling and the Computer (Studies in Language Linguistics). Addison-Wesley Longman Ltd, dec 1995. ISBN 0582234794.

[37] Levenshtein, V.I., 1966, February. Binary codes capable of correcting deletions, insertions, and reversals. In Soviet physics doklady (Vol. 10, No. 8, pp. 707-710).

[38] Wagner, R.A. and Fischer, M.J., 1974. The string-to-string correction problem. Journal of the ACM (JACM), 21(1), pp.168-173.

[39] Hamming, R.W., 1950. Error detecting and error correcting codes. The Bell system technical journal, 29(2), pp.147-160.

[40] Pilar Angeles, M. del a Espino Gamez, A. Comparison of methods Hamming Distance, Jaro, and Monge-Elkan. In: DBKDA 2015: the seventh international conference on advances in databases, knowl-edge and data applications. 2015.

[41] Damerau, F.J., 1964. A technique for computer detection and correction of spelling errors. Communications of the ACM, 7(3), pp.171-176.

[42] Winkler, William E. "String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage." (1990).

[43] Mitton., R., Birkbeck spelling error corpus, 1985. http://ota.ahds.ac.uk/ (Last accessed: March 2007).

[44] TshwaneDJe HLT. Sesotho sa leboa corpora, 2006. Private Communication.September 2006.

[45] Rudy, R. and Naga, D.S., 2018. Fast and Accurate Spelling Correction Using Trie and Damerau-levenshtein Distance Bigram. TELKOMNIKA (Telecommunication Computing Electronics and Control), 16(2), pp.827-833.

[46] Cordewener, K.A., Verhoeven, L. and Bosman, A.M., 2016. Improving spelling performance and spelling consciousness. The journal of experimental education, 84(1), pp.48-74.

[47] Van Zaanen, M. and Van Huyssteen, G., 2003. Improving a spelling checker for Afrikaans. In Computational Linguistics in the Netherlands 2002 (pp. 143-156). Brill Rodopi.

[48] Huang, Y., Murphey, Y.L. and Ge, Y., 2013, April. Automotive diagnosis typo correction using domain knowledge and machine learning. In 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM) (pp. 267-274). IEEE.

[49] Etoori, P., Chinnakotla, M. and Mamidi, R., 2018, July. Automatic spelling correction for resource-scarce languages using deep learning. In Proceedings of ACL 2018, Student Research Workshop (pp. 146-152).

[50] Carlson, A. and Fette, I., 2007, December. Memory-based context-sensitive spelling correction at web scale. In Sixth International Conference on Machine Learning and Applications (pp. 166-171). IEEE.

[51] Hussain, S. and Naseem, T., 2013. Spell checking. Crulp, Nuces, Pakistan, www. crulp. org.

[52] Dhakal, V., Feit, A.M., Kristensson, P.O. and Oulasvirta, A., 2018, April. Observations on typing from 136 million keystrokes. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (pp. 1-12).

[53] Rimbar, H., 2017. The Influence of Spell-checkers on Students' ability to Generate Repairs of Spelling Errors. Journal of Nusantara Studies (JONUS), 2(1), pp.1-12.

[54] Hofstede, R., Čeleda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A. and Pras, A., 2014. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. IEEE Communications Surveys & Tutorials, 16(4), pp.2037-2064.

[55] Holkovič, M., Polčák, L. and Ryšavý, O., 2019, July. Application Error Detection in Networks by Protocol Behavior Model. In International Conference on E-Business and Telecommunications (pp. 3-28). Springer, Cham.

[56] Moore, T. and Edelman, B., 2010, January. Measuring the perpetrators and funders of typosquatting. In International Conference on Financial Cryptography and Data Security (pp. 175-191). Springer, Berlin, Heidelberg.

[57] Grudin, J.T., 1983. Error patterns in novice and skilled transcription typing. In Cognitive aspects of skilled typewriting (pp. 121-143). Springer, New York, NY.

A.8 PCAPFunnel: A Tool for Rapid Exploration of Packet Capture Files

Authors: Uhlár Juraj, Ing. (40%), Holkovič Martin, Ing. (30%), Rusňák Vít, RNDr., Ph.D. (30%)
Abstract: Analyzing network traffic is one of the fundamental tasks in both network operations and security incident analysis. Despite the immense efforts in workflow automation, an ample portion of the work still relies on manual data exploration and analytical insights by domain specialists. Current state-of-the-art network analysis tools provide high flexibility at the expense of usability and have a steep learning curve. Recent - often web-based - analytical tools emphasize interactive visualizations and provide simple user interfaces but only limited analytical support. This paper describes the tool that supports the analytical work of network and security operators. We introduce typical user tasks and requirements. We also present the filtering funnel metaphor for exploring packet capture (PCAP) files through visualizations of linked filter steps. We have created PCAPFunnel, a novel tool that improves the user experience and speeds up packet capture data analysis. The tool provides an overview of the communication, intuitive data filtering, and details of individual network nodes and connections between them. The qualitative usability study with nine domain experts confirmed the usability and usefulness of our approach for the initial data exploration in a wide range of tasks and usage scenarios, from educational purposes to exploratory network data analysis.
Keywords: Data analysis, Data visualization, Network traffic analysis, Packet captures.
Published in: 25th International Conference Information Visualisation (IV 2021), Sydney, Australia
Conference rating: B(Core), B1(Qualis)
ISBN: 978-1-6654-3827-8

PCAPFunnel: A Tool for Rapid Exploration of Packet Capture Files

Juraj Uhlár
Flowmon Networks
Brno, Czech Republic
Email: juraj.uhlar@flowmon.com

Martin Holkovič
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
Email: iholkovic@fit.vutbr.cz

Vít Rusňák
Institute of Computer Science
Masaryk University
Brno, Czech Republic
Email: rusnak@ics.muni.cz

***Abstract*—Analyzing network traffic is one of the fundamental tasks in both network operations and security incident analysis. Despite the immense efforts in workflow automation, an ample portion of the work still relies on manual data exploration and analytical insights by domain specialists. Current state-of-the-art network analysis tools provide high flexibility at the expense of usability and have a steep learning curve. Recent—often web-based—analytical tools emphasize interactive visualizations and provide simple user interfaces but only limited analytical support. This paper describes the tool that supports the analytical work of network and security operators. We introduce typical user tasks and requirements. We also present the filtering funnel metaphor for exploring packet capture (PCAP) files through visualizations of linked filter steps. We have created PCAPFunnel, a novel tool that improves the user experience and speeds up packet capture data analysis. The tool provides an overview of the communication, intuitive data filtering, and details of individual network nodes and connections between them. The qualitative usability study with nine domain experts confirmed the usability and usefulness of our approach for the initial data exploration in a wide range of tasks and usage scenarios, from educational purposes to exploratory network data analysis.**

***Index Terms*—Data analysis, Data visualization, Network traffic analysis, Packet captures**

I. Introduction

Day-to-day monitoring of the network status and intervention, if necessary, is part of the network administrators' duties [1]. Many problems appear, causing the network or its parts to malfunction daily. E.g., communication issues with the webserver, broken connections between distant locations, or a user's computer spreading malware across the network. Except for real-time monitoring, packet capture inspection is a standard method used during network analysis. However, it is nearly impossible to analyze all traffic on a per-packet basis due to the overwhelming amount of transferred data, especially in high-speed networks. Therefore the network administrators work with the aggregated data only, which yields to the crucial challenge: selecting the proper criteria for data aggregation and their presentation. They need to explore multi-dimensional data and gain knowledge by analyzing them on multiple levels of abstraction [2].

Commonly used low-level command-line tools or applications like Wireshark [3] provide broad data processing and filtering capabilities but provide limited data visualization and presentation in a comprehensible way. The previous studies confirmed the usefulness of visualizations for network analysis [4], [5] which outperforms the traditional text-based ones. Novel tools such as NetCapVis [6] bootstrap the network analysis through interactive and easy-to-learn graphical interfaces, which makes them accessible also for novice users. However, less attention is on the support and guidance of the analysts in their exploratory work.

We address these issues in the prototype implementation of a web-based visual analysis tool for packet capture (PCAP) data. *PCAPFunnel* enables to upload packet capture files and performs a rapid exploration through a set of consecutive filters. It improves the initial orientation in the network dataset and allows the export and import of filter configurations to simplify their sharing or reuse. The filtering funnel metaphor guides the analysts and allows quick drill down to the details of individual network nodes or connections between them. We performed a qualitative user evaluation with nine domain experts who considered our approach helpful in a wide range of usage scenarios for exploratory network data analysis or educational purposes. The project has been done in collaboration with Flowmon Networks a.s., a company providing network monitoring solutions.

We contribute to state of the art with: 1) a filtering funnel metaphor applied on an exploration of network traffic datasets, 2) design and implementation of PCAPFunnel, a visual analysis tool for rapid exploration of PCAP files,

Section II covers the related work in network traffic analysis. Section III provides the data abstraction, outlines user tasks, and presents design requirements on analytical tools for our target domain. In Section IV, we describe PCAPFunnel design. Section V summarizes the qualitative user study demonstrating the usability and usefulness of our approach. We present the discussion and outline future work in Section VI and conclude our paper in Section VII.

II. Related Work

We first introduce the network traffic data sources. To better situate our work, we present the three-level workflow of network traffic analysis, including examples of currently

used tools. Finally, we discuss existing visualization tools that inform the PCAPFunnel design or provide akin capabilities.

A. Network Traffic Data Sources

The network traffic analysis can be either active or passive [7]. The active approach requires generating additional network traffic to check a device's status or link. The passive approach uses only data that are already available in the network. In the remainder, we focus on passive analysis as it is more accessible since it does not require any additional network infrastructure.

Two primary data sources for passive network traffic analysis are packet captures (PCAP) and network flows (NetFlow). A PCAP file [8] contains complete copies of packets transferred over the network. Therefore, it allows analyzing all values from packet headers and payloads. Because storing and processing complete packets is very expensive, it is usually not performed globally but only for a limited network segment. NetFlow [9] represents connections between communicating nodes (i.e., flows). Compared to PCAP, it contains only a limited amount of attributes from each packet. While NetFlow technology is predominantly used in real-time network monitoring, PCAPs are suitable for detailed inspection of network performance issues or forensic analysis of detected cybersecurity incidents [10].

B. Network Traffic Analysis

There are three levels network traffic analysis that differ in the level of automation and depth of the required knowledge of the analysts or network operators who perform them:

Automatic diagnostic tools process the captured network traffic, try to identify network issues, and report them [7], [11], [12]. A common strategy is that the administrators validate these reports and take countermeasures if necessary. However, the diagnostic tools heavily rely on the knowledge base they implement, which often limits their capabilities. Although most of the routine tasks are automated, many reported network issues require administrators' attention. An example of such a tool is Flowmon Packet Investigator [13].

The **top-level analysis** is usually performed over the aggregated data. It includes data sorting and filtering according to several different criteria. Although the workflow is straightforward, it gives the administrators the power to explore even massive network traffic and identify context before diving into the details. One of the tools for NetFlow data analysis is NFDump [14].

The **in-depth analysis** allows the administrators to explore the detailed information and content of individual packets [15]. It is powerful but time-consuming and often also prone to losing the "big picture." Therefore, the analysts often switch continually between the top-level and in-depth analyses in their work.

To capture and inspect the traffic, network administrators and security analysts often use command-line utilities (e.g., tcpdump, TShark) or highly flexible tools, such as Wireshark [3] and NetworkMiner [16]. The latter ones also provide statistics as well as details of each captured packet. However, they present the data in tables offering only limited visualization capabilities (e.g., tabular row highlighting or simple static line or bar charts). They are, however, used mainly by skilled users, and the steep learning curve is one of their main disadvantages.

These three approaches are usually combined into a complex workflow of network or security operations (NetOps or SecOps) teams. The automatic diagnostic tools notify the administrators about the network issue, and they initiate the top-level analysis. If these two steps do not provide sufficient information, the analysts continue with in-depth network traffic inspections using proper tools [17].

In our work, we focus wholly on top-level analysis. PCAPFunnel aims to support the analysis bootstrap, reduce the *time-to-first-insight*, and improve the analytical process by providing better guidance and support through interactive visualizations.

C. Packet Capture Visualizations

With the growing amount of transferred network data, the design of efficient visualizations gains importance [18]. Visualizations are gradually used in various areas of network security [19]. Over a decade ago, Goodall [5] performed a comparative evaluation of a conventional PCAP file analysis tool with a visualization application. The visualization application outperformed the conventional tool in both task accuracy and completion times. The evaluation also provided insights and the overall preference of study participants. Further, we show several examples of such visualization tools.

GrassMarlin [20] is an open-source tool released by the NSA. Its primary purpose is to help passively map industrial control systems and networks and visualize them in a communication graph.

SNAPS [21] and EventPad [22] represent specialized tools for network analysts and require strong domain knowledge. Both the tools work with PCAP files. The former provides a bottom-up pixel-oriented approach for iterative analysis and parallel filtering options. The latter enables pattern identification and analysis of malware activity using visual analytics methods. Another example of an advanced visualization tool is FloVis [23], a suite of visualization tools intended to complement command-line utilities. It processes NetFlow data and visualizes activity diagrams, communication clustering, and connection details.

The advances in modern web application development open new possibilities for leveraging visual analytics methods in network analysis tools. Several tools provide analytical capabilities as a service. For example, CapAnalysis [24] allows users to review large PCAP files, parse the data streams, filter out ports, protocols, or IP addresses, and associate them with geographical areas. A-packets [25] and PacketTotal [26] provide multiple individual views on PCAP files. Rather than supporting the explorative analysis, both tools provide dataset overview and multiple dashboards focused on individual characteristics extracted from the data (e.g., application, SSL certificates, transferred files). The closest to our approach is NetCapVis [6]. It provides both overview and fundamental analytical support through filtering based on incoming and outgoing IP addresses and port numbers. Its main limitation is only limited details

for individual network nodes or connections between them. However, a user can export filter configuration for Wireshark to enable investigation of further details.

In PCAPFunnel, though inspired by these tools, we focused on improving the support and guidance to reduce users' cognitive load during the analytical process. Our goal was to design a tool that will enable rapid initial exploration of the dataset, be useful for skilled professionals, and easy to learn for novice users.

III. Data, Tasks and Requirements

The section provides the data abstraction of PCAP files, followed by an overview of the user tasks. Based on these, we formulate five design requirements for PCAPFunnel.

A. Data Abstraction

PCAP is a binary file format described in [8]. The file structure starts with a global header followed by at least one record for each packet consisting of a packet header and its payload (i.e., content data). There are multiple implementations (e.g., WinPcap, NPcap) that differ in API methods and supported features. Though, we worked with the most common *libpcap*[1], considered a de-facto standard. Moreover, we currently use only a subset of PCAP data needed for the visualizations.

We represent each packet as an object with several attributes listed in Table I to simplify data manipulation.

We also enhanced the packet with the *application name* attribute representing the application or service corresponding to the source or destination port. The information is based on the IANA Registry [27].

TABLE I
List of packet attributes used in PCAPFunnel.

Attribute	PCAP Data Property
index	`frame.number`
timestamp	`frame.time_epoch`
network/transport protocol	`frame.protocols`
source/destination IP	`src/dst` (e.g., `ip.src`, `ip.dst`)
source/destination port	`srcport/dstport` (e.g., `udp.srcport`)
bytes	`frame.len`

B. Tasks

Although NetOps and SecOps teams focus on different goals, they perform similar network analysis tasks on the same input data. Ulmer et al. [6] identified three general tasks performed by both teams: *GT1:* Get a network traffic overview. *GT2:* Find suspicious connections. *GT3:* Determine relevant events for an in-depth analysis.

Based on discussion with domain experts, we have identified five specific tasks (ST), also common to both teams:

ST1: Identify the top N communication sources, based on given criteria;

ST2: Discover unusual patterns in the network traffic (e.g., peaks);

ST3: Identify nodes with which the particular station communicated.

ST4: Identify nodes providing specific services to a network (e.g., DNS server);

ST5: Share the analysis parameters with coworkers.

We derived these tasks from real-world scenarios and the personal experience of several Flowmon Networks employees.

C. User Requirements

We address the user tasks through six design requirements that have driven our work on PCAPFunnel:

R1: Provide descriptive statistics of the network's traffic properties (e.g., traffic volume, number of connections, top N statistics for a typical network (e.g., IP, ICMP) and transport (e.g., UDP, TCP) protocols, ports, or IP addresses).

R2: Enable intuitive and progressive filtering of data by multiple packet properties.

R3: Provide details for individual nodes and connections between them and identify individual packets for further analysis in external packet analyzer software.

R4: Enhance the PCAP data with information from external sources where possible (e.g., network node geolocation, DNS resolving).

R5: Support sharing and reuse of filter configurations.

R6: Allow working with large datasets progressively when loaded without disrupting the user experience.

IV. PCAPFunnel Design

PCAPFunnel is a React-based web application communicating with a Node.js server and external APIs, providing complementary information about individual IP addresses (e.g., resolved DNS names and geolocations). The demo is available at https://pcap-viz.surge.sh.

The main goals of the tool are (a) to reduce the *time-to-first-insight* and (b) to support the users in their explorative drill down of PCAP files. The application follows Shneiderman's "Overview first, zoom and filter, then details on demand" [28]. For data organization and separating different detail levels, the application leverages tabs. The Dataset Overview provides filtering options and an overview of the loaded PCAP dataset. Detailed Views display statistics and attributes of network entities: network nodes (defined by their IP addresses) or connections between them. In the following sections, we discuss these views in detail.

A. Dataset Overview

The user interface has two main sections. The left part (Fig. 1(a–e)) serves for data input and filtering, the right one (Fig. 1(f))for data visualization and exploration (**R1**).

File Upload and Summary:

A user can upload a PCAP file from local storage or as a publicly available URL (Fig. 1(a)). The uploaded PCAP file is split into progressively processed batches. The packet attributes from each batch are extracted using TShark and stored as a JSON file on the server. Fig. 2 overviews the data preprocessing workflow.

This approach allows the user to start exploring the data while the PCAP file is still being processed (**R6**). The dataset

[1] https://www.tcpdump.org

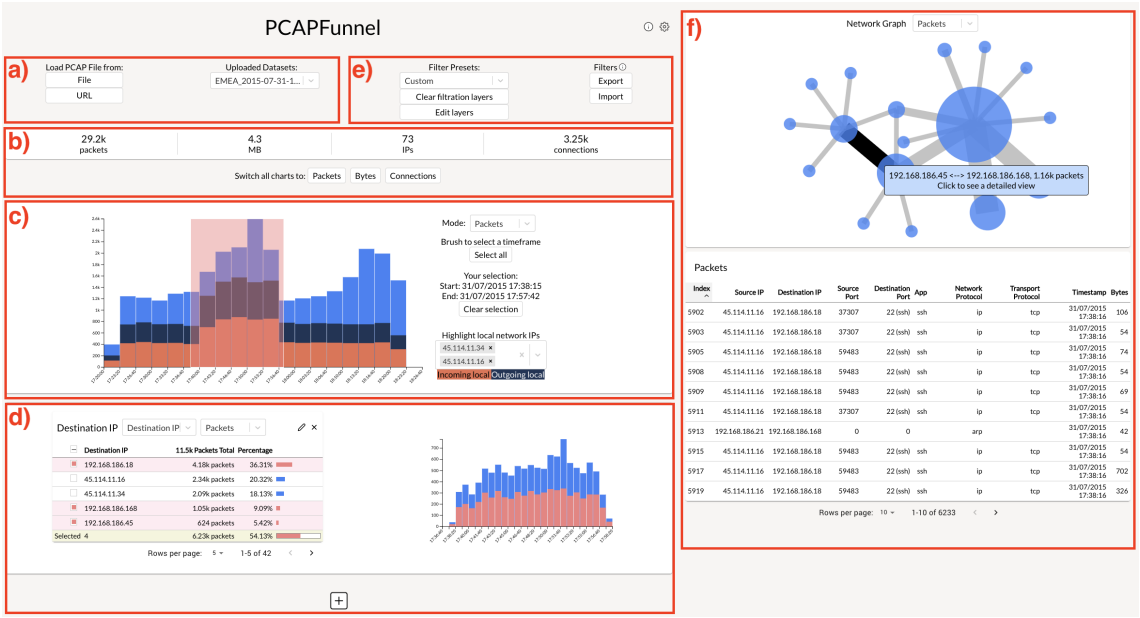

Fig. 1. DATASET OVERVIEW visualizes network traffic from packet capture files: (a) One can upload a PCAP file or choose from the previously uploaded datasets (a). Dataset summary (b) shows the sum of packets, IP addresses, connections, and dataset size. Communication profile (c) is the starting point for the analysis. The user can progressively configure up to six filter steps (d) to filter out the data according to multiple consecutive attributes. Users can also export and import the filter configurations or use one of four predefined presets (e). The right part (f)shows the filtering results as a network graph and in tabular form.

summary—Fig. 1(b)—provides an overview of the loaded dataset parameters: a sum of packets, IP addresses, connections and the file size.

Filtering Funnel: The network traffic analysis often requires a combination of multiple filter steps. A common approach is, therefore, to chain multiple filters into one filtering command. We implemented such chained filters using the *filtering funnel metaphor*, a fundamental concept of PCAPFunnel.

PCAP file analysis starts with a statistical overview of traffic properties (e.g., traffic volume, peaks in communication, primary sources or used protocols, and distribution). Filtering is then based on a chain of filter steps where each step allows to filter data using a different property. The output of the preceding filter step becomes the input of the following one. So the data is progressively filtered, resembling a funnel as depicted in Fig. 3. Depending on the analytical goal, the steps can be combined almost arbitrarily from top-down or bottom-up manner, i.e., from application to network-level or vice versa (**R2**).

The traffic volume over time is displayed as a bar chart. The chart also serves as the first filter step, allowing the user to select the time range (Fig. 1(c)). Only the traffic information from the selection is passed to the filter steps below. The user can also indicate local network IP addresses by selecting them in the drop-down dialog.

The filter step (Fig. 1(d)) consists of a table and the traffic volume chart. The table shows packet property statistics (e.g.,

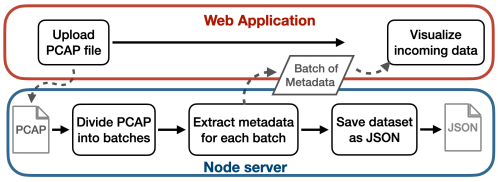

Fig. 2. PCAP file data preprocessing workflow.

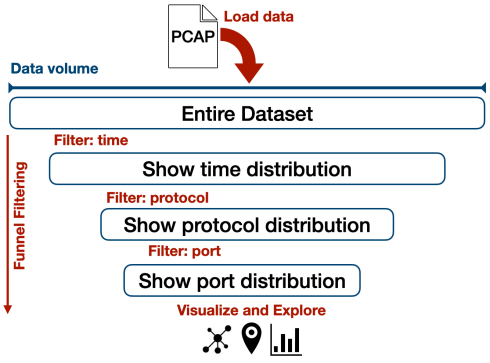

Fig. 3. The *filtering funnel metaphor* illustration. Step-by-step filtering allows to drill down in the data progressively.

a destination IP filter contains a table of all the IP addresses from the selection). The [+] button allows to add up to six different filter steps corresponding to the following packet properties: IP address (source, destination, or both), network protocol, transport protocol, port (source, destination, or both), and application name. The applied filter steps remain visible. Since they are linked, the change in one instantly affects others.

In addition to an arbitrary combination of filter steps, the user can select from four presets and export or import filtering configurations (Fig. 1(e)). There are two presets for a top-down analysis (i.e., from an application to an IP address) and two in reversed order. The exported files are in JSON format to simplify their sharing with co-workers or reuse on similar datasets (**R5**).

Filtering Results: Communications that pass through all the steps are displayed in the DATASET OVERVIEW right part as a node-link diagram (Fig. 1(f)). The table below lists all the packets and their attributes. Any changes in filtering steps directly affect the displayed results.

The node-link diagram visualizes the communication topology of the filtered data. Each node corresponds to an IP address. Its size is proportional to the volume of send and received data. The link width represents the volume of communication between two nodes.

Clicking on a node or table row opens a new detailed view tab with information concerning the given IP address. Analogically, clicking on a link opens a detailed view tab of the traffic between the two IP addresses. The user can also switch between visualizing packets, bytes, or connections (uni- or bi-directional).

B. Detail View

The DETAIL VIEW (Fig. 4) provides further details of incoming and outgoing communication of the selected IP address or addresses in case of connections (**R3**). The view has four sections: *Communication Profile*, *Network Profile*, *Packet Property Statistics*, and *Raw Data*.

Communications profile: The section contains four mirrored charts, each displaying incoming (top) and outgoing (bottom) traffic (Fig. 4(a)). The charts display the traffic volume over time in numbers of packets, bytes, and flows. The last one is a packet size histogram.

Network Profile: The section (Fig. 4(b)) visualizes the proportional comparison of incoming and outgoing traffic, geolocalized IP addresses, and the list of the countries based on the traffic volume. The geolocalization (**R4**) is provided through GeoLite2 IP [29].

A Sankey diagram displays incoming (left) and outgoing (right) connections based on the number of packets, bytes, or flows. The same color signifies the same source/destination IP. The user can also display only the top 5, 10, 15, 20, or 30 connections. "Others" aggregates the remaining ones.

The minimap shows all geo-localized IP addresses. If both connection endpoints are localized, there is also a link between them. Circle size represents an IP address and is proportional to the sum of incoming and outgoing traffic.

The table on the right side lists the top countries by traffic volume. The user can also switch between the table and choropleth visualization.

All the visualizations are zoomable and draggable. Clicking on a node or connection either in the Sankey diagram or geovisualization opens a new Detailed View tab with the corresponding IP address or connection. So the user can continue further in the drill-down analysis.

Packet Property Statistics: The third section (Fig. 4(c)) presents the most frequent (i.e., top 5/10/15/20/30) values for packet properties: source and destination IP addresses, network protocol, transport protocol, source and destination ports, and the application name. Each property data is visualized in one of seven bar charts.

Raw Data: The last section (Fig. 4(d)) presents the raw data in two tables: *Connections* and *Packets*.

The *Connections* table lists collections of packets with the same transport protocol, source IP and port, destination IP, and port. The table also displays the first packet's timestamp and overall packet count in the connection. The user can switch between uni-directional and bi-directional connections.

A country flag is shown in the source and destination IP columns for localized IP addresses. Another enhancement is DNS translation which converts IP addresses to resolved domain names (**R4**). IP addresses and hostnames are clickable and open new DETAIL VIEW tab of the clicked IP address. The *Packets* table displays the attributes of corresponding packets that provide the underlying information for all the visualizations in the DETAIL VIEW. Both tables are sortable by clicking on the column headings. A click on the IP address invokes a new DETAIL VIEW tab with corresponding data.

V. USER STUDY

We conducted a qualitative user study of PCAPFunnel with the following goals: a) collect qualitative feedback on the usability and usefulness, b) identify limitations of the tool, and c) assess the fulfillment of the user requirements. Instead of formal usability evaluation or measuring task performance, we sought to observe what aspects of the tool provide the most value to domain experts. The approach is commonly used in projects like ours [30], [31]. Due to Covid-19 pandemic restrictions, we held the study online.

A. Method

Participants: We recruited nine domain experts, all males (26–42 yo). Three of them work in academia as cybersecurity researchers. Two are managers, and four employees in the private sector. All participants have previous experience with computer network security (six over ten years, two 5–10 years, one less than five years). Six participants participated over video calls (Google Meet) which were also recorded. Three (P7–P9) worked asynchronously due to their time restrictions. In both cases, the study design was equal. Measured on a five-point Likert scale (1=novice, 5=expert), all the participants considered themselves as experienced with packet analysis (mean 3.8) and Wireshark (mean 3.7). The participants had

Fig. 4. DETAIL VIEW showing communication details of the IP address 192.168.15.4.

also experience with other PCAP analysis tools such as TShark, tcpdump, Moloch, SELK, pyshark, Microsoft Network Monitor, or Netfox Detective.

Procedure and Tasks: We prepared a Google Form that served as the evaluation guide. The form also included the prerecorded video presenting PCAPFunnel features, description of the tasks, and demographic and post-study questionnaires. The three participants who performed the study asynchronously used the form to list the steps they performed to complete each task and for written feedback. Despite being recorded, their responses still provided helpful feedback, and therefore we included them in the evaluation results.

The procedure has three phases: introduction phase, task performing phase, and debriefing phase. In the introduction phase, the participants consented and filled the demographic questionnaire. Next, they watched the presentation of PCAPFunnel features and have a couple of minutes to interact with the tool using a dummy dataset.

The task performing phase (∼40 minutes) consisted of five tasks:

Task 1: Identify the application with the most significant number of connections.

Task 2: Identify two 2 IP addresses that are the most significant sources of the most prominent peak in the traffic.

Task 3: Given the IP address of an infected network station, find out which country the station has communicated with using the HTTPS application protocol.

Task 4: Which network nodes provide DNS service on other ports than 53?

Task 5: Import the provided filter configuration. What is the location of the destination IP address in the resulting connection?

We developed specific tasks rather than open-ended exploration, so the participants tried most PCAPFunnel user interface features. Most of the tasks, however, were possible to accomplish in several ways. Each task was introduced by a brief description providing the context and preparation steps that included, e.g., loading a new dataset for a given task. We asked participants of video calls to think aloud [32] and to rate the perceived difficulty.

In the debriefing phase (10–15 minutes), the participants filled closed-question post-study questionnaire and shared their suggestions and opinions on our tool.

The participants used their computers. Their screen resolutions were FullHD (6×), QHD (2×), or UHD (1×). Since the PCAPFunnel is a web application, they used either recent Google Chrome (7×) or Firefox (2×) browsers.

B. Results

The participants provided helpful feedback about the PCAPFunnel design. Overall, the results are encouraging in the usability and usefulness of the tool.

Tasks: As we can see from Table II, the participants were largely successful when solving the tasks and considered them relatively easy.

Overall, the participants engaged well with the tasks, and we neither received any reservations regarding their purpose or realism level. The participants solved tasks without the interventions of the observers, so the perceived task difficulty was not necessarily correlated with its success rate. Tasks 1 and 5 achieved high success rates and low perceived difficulty. The participants perceived Task 2 as the easiest, despite most of

TABLE II
PERCEIVED TASK DIFFICULTY (7-POINT LIKERT SCALE—HIGHER IS EASIER) AND SUCCESS RATE (RED CELLS INDICATE FAILED TASKS).

Participant	Task 1	Task 2	Task 3	Task 4	Task 5
P1	6	6	4	7	7
P2	7	6	5	5	7
P3	1	7	6	1	7
P4	6	6	5	6	7
P5	6	7	7	7	7
P6	2	6	1	4	7
P7	6	6	1	6	3
P8	5	6	4	7	3
P9	7	7	6	7	7
Mean success rate	89%	44%	89%	66%	100%
Mean task difficulty	5.1	6.3	4.3	5.6	6.1

them submitted only partial answers. Typically, the participants forgot to change one or more filter steps (e.g., they forgot to switch to connections instead of packets), resulting in a slightly different set of resulting IP addresses.

The participants P6 and P7 considered Task 3 as very difficult since they got confused with the choropleth visualization in the DETAIL VIEW. Based on this feedback, we favored table view as the primary and choropleth as the secondary option.

Some confusion arose around the correct interpretation of Task 4, resulting in a lower success rate. It was caused by misinterpretation of the task. Since we revealed the task assignment flaw (there were two possible interpretations) after a couple of sessions, we decided to finish the rest without the change.

Usability and Usefulness: The participants worked with PCAPFunnel without major issues after being given a brief introduction and quickly grasped the filtering funnel metaphor principle.

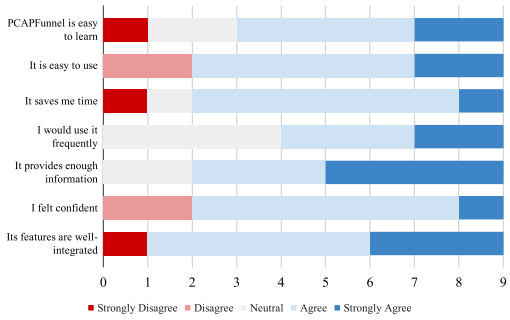

Fig. 5. Answers from the post-study questionnaire.

They were mostly positive (Fig. 5) about its features and perceived it as easy to learn (mean 3.7) and use (mean 3.8). Most of them felt confident when solving the tasks. However, some participants expressed the need for more time to become acquainted (P3: "I would need some more practice with the tool"). Most of them think that PCAPFunnel provides them with enough information (mean 4.2) and that its features are well-integrated (mean 4.0). P4 remarked that "comparing to other tools he uses is [PCAPFunnel] very intuitive." Most of them also indicated they would like to use the application more often (mean 3.8) since it could save their time (mean 3.7). P2 also remarked that "[he] likes its minimalistic design."

VI. DISCUSSION AND FUTURE WORK

The usability study confirmed our initial assumptions that our approach to multi-step filtering could improve the user experience of the top-level PCAP analysis process. Our observations from the study provide several implications for future research.

Desired Features and Improvements: The majority of participants considered the tool as well-designed. However, we also received several suggestions on improvements, some of which we have already integrated into the final design—for example, opening new DETAIL VIEW either by clicking on the table row or in visualizations. We also revealed and fixed few minor bugs in the implementation and slightly changed the terminology used for several visualizations.

Few participants often expressed the need for more filtering capabilities, such as allowing negative filters by using the logical NOT operator (P3) or the possibility to enter the filter as a text input (P5). Additional configuration features include optional DNS resolving (P1) or anonymization of imported data (P2).

The truth is that we intentionally omitted many of these features to keep the prototype compact, but we plan to integrate them in the next versions of the tool.

Using PCAPFunnel for Training and Education: Few participants remarked that the tool is so easy to work with that it could be used during the network administration training. As we already mentioned, current tools for network analysis usually provide only limited visualization capabilities and are hard to learn. PCAPFunnel could support teachers in explaining various phenomena, identifying communication patterns, or demonstrating network analysis tasks during the classes. Moreover, the use of familiar visualizations makes the tool also suitable for novice users.

Generalizability: Currently, our prototype works only with PCAP files. However, the filtering funnel metaphor is generic and can be used with other data sources (e.g., NetFlow or logs). We would also like to extend export formats to PDF files to document or present the analysis results. However, making PCAPFunnel part of the analysts' toolkit would require its integration with other deep packet inspection tools. The trivial way is to create new filter export in a format comprehensible for Wireshark or similar applications. The more ambitious way is to provide integration with deep packet inspection tools and extend the PCAPFunnel user interface, so the user does not need to switch the tools. Such integration would allow seamless transition from top-level to in-depth analysis and back.

Performance Improvements: One of the current weaknesses of PCAPFunnel is the need to upload the entire PCAP file to the server before the progressive data processing starts. We plan to improve our preprocessing to start as soon as the first packets are uploaded, similarly to the approach used by NetCapVis [6].

VII. Conclusion

The traditional network traffic analysis tools have a steep learning curve and only limited visualization capabilities. To facilitate and speed up the process, we designed PCAPFunnel—a tool for packet capture analysis. Inspired by previous work and based on tight collaboration with domain experts, we defined user tasks and design requirements.

We proposed the *filtering funnel metaphor* to support filtering and data analysis based on linking several independent filter steps where the output from one is another's input. All the filter steps are permanently visible to the user, who can interactively modify filters' parameters. With this new approach, a domain expert can quickly check the data's content, determine its structure, analyze network actors' behavior, or reveal the cause of network issues. The user interface leverages linked views and conventional visualizations, so it is also suitable for novice users.

We further performed the user study with nine domain experts in the field of network data analysis. The participants appreciated that PCAPFunnel is easy to learn and use and considered the proposed method flexible and supportive during the initial packet capture data analysis. Likewise similar user studies, we admit that even our study has several limitations. Namely, a low number of participants and its qualitative focus. A more extensive deployment in a real-world setup could provide new insights. Also, a comparative study with the commonly used tools could be valuable. We hope that PCAPFunnel will also inspire novel approaches for network traffic analysis based on interactive visualizations.

Acknowledgment

This research was supported by ERDF "CyberSecurity, CyberCrime and Critical Information Infrastructures Center of Excellence" (No. CZ.02.1.01/0.0/0.0/16_019/0000822).

References

[1] R. Wang *et al.*, "Knight's Tour-based Fast Fault Localization Mechanism in Mesh Optical Communication Networks," *Photonic Network Communications*, vol. 23, no. 2, pp. 123–129, 2012.

[2] R. Ball, G. A. Fink, and C. North, "Home-centric Visualization of Network Traffic for Security Administration," in *International Workshop on Visualization for Cyber Security*, C. E. Brodley, P. Chan, R. Lippmann, and W. Yurcik, Eds. ACM, 2004, pp. 55–64.

[3] G. Combs *et al.*, "Wireshark: A Network Protocol Analyzer," 2008. [Online]. Available: http://www.wireshark.org/

[4] E. W. Bethel, S. Campbell, E. Dart, K. Stockinger, and K. Wu, "Accelerating Network Traffic Analytics Using Query-Driven Visualization," in *2006 IEEE Symposium On Visual Analytics Science And Technology*, 2006, pp. 115–122.

[5] J. R. Goodall, "Visualization is Better! A Comparative Evaluation," in *6th International Workshop on Visualization for Cyber Security*, 2009, pp. 57–68.

[6] A. Ulmer, D. Sessler, and J. Kohlhammer, "NetCapVis: Web-based Progressive Visual Analytics for Network Packet Captures," in *2019 IEEE Symposium on Visualization for Cyber Security (VizSec)*, 2019, pp. 1–10.

[7] M. łgorzata Steinder and A. S. Sethi, "A Survey of Fault Localization Techniques in Computer Networks," *Science of computer programming*, vol. 53, no. 2, pp. 165–194, 2004.

[8] G. Harris and M. Richardson, "PCAP Capture File Format," Working Draft, IETF Secretariat, Internet-Draft draft-gharris-opsawg-pcap-01, December 2020. [Online]. Available: http://www.ietf.org/internet-drafts/draft-gharris-opsawg-pcap-01.txt

[9] B. Claise, "Cisco Systems NetFlow Services Export Version 9," Internet Requests for Comments, RFC Editor, RFC 3954, October 2004. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3954.txt

[10] G. A. Pimenta Rodrigues *et al.*, "Cybersecurity and Network Forensics: Analysis of Malicious Traffic Towards a Honeynet with Deep Packet Inspection," *Applied Sciences*, vol. 7, no. 10, p. 1082, 2017.

[11] C. Guo *et al.*, "Pingmesh: A large-scale System for Data Center Network Latency Measurement and Analysis," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 139–152.

[12] M. Holkovič and O. Ryšavý, "Using Rule-Based Decision Trees for Automatic Passive Diagnostics of the Network Problems," *International Journal on Advances in Networks and Services*, vol. 2020, no. 1, pp. 1–10, 2020.

[13] Flowmon Networks a.s., "Flowmon Packet Investigator," 2020. [Online]. Available: https://www.flowmon.com/cs/products/software-modules/packet-investigator

[14] P. Haag, "Nfdump – netflow processing tool," 2007. [Online]. Available: https://github.com/phaag/nfdump

[15] V. Ndatinya *et al.*, "Network Forensics Analysis Using Wireshark," *International Journal of Security and Networks*, vol. 10, no. 2, pp. 91–106, 2015.

[16] Netresec AB, "NetworkMiner." [Online]. Available: https://www.netresec.com/?page=networkminer

[17] M. Solé *et al.*, "Survey on Models and Techniques for Root-cause Analysis," *arXiv preprint arXiv:1701.08546*, 2017.

[18] H. Shiravi, A. Shiravi, and A. A. Ghorbani, "A Survey of Visualization Systems for Network Security," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 8, pp. 1313–1329, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/tvcg/tvcg18.html#ShiraviSG12

[19] H. Tang, C. Han, and J. Ge, "Applications of Visualization Technology for Network Security," in *TrustCom/BigDataSE/ICESS*. IEEE Computer Society, 2017, pp. 1038–1042. [Online]. Available: http://dblp.uni-trier.de/db/conf/trustcom/trustcom2017.html#TangHG17

[20] NSA, "GRASSMARLIN," 2017. [Online]. Available: https://github.com/nsacyber/GRASSMARLIN

[21] B. C. M. Cappers and J. J. van Wijk, "SNAPS: Semantic Network Traffic Analysis Through Projection and Selection," in *2015 IEEE Symposium on Visualization for Cyber Security (VizSec)*, L. Harrison, N. Prigent, S. Engle, and D. M. Best, Eds. IEEE Computer Society, 2015, pp. 1–8.

[22] B. C. M. Cappers *et al.*, "Eventpad: Rapid Malware Analysis and Reverse Engineering using Visual Analytics," in *2018 IEEE Symposium on Visualization for Cyber Security (VizSec)*. IEEE, 2018, pp. 1–8.

[23] T. Taylor, D. Paterson, J. Glanfield, C. Gates, S. Brooks, and J. McHugh, "FloVis: Flow Visualization System," in *2009 Cybersecurity Applications Technology Conference for Homeland Security*, 2009, pp. 186–198.

[24] Costa, Gianluca, "CapAnalysis," 2018. [Online]. Available: http://www.capanalysis.net/

[25] A-Packets, "Online PCAP file analyzer designed to visualize HTTP, Telnet, FTP," 2020. [Online]. Available: https://apackets.com/

[26] PacketTotal, "Simple, free, high-quality PCAP analysis." [Online]. Available: https://packettotal.com/

[27] "Service Name and Transport Protocol Port Number Registry." [Online]. Available: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

[28] B. Shneiderman, "The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations," in *Proceedings 1996 IEEE Symposium on Visual Languages*, 1996, pp. 336–343.

[29] "GeoLite2 Free Downloadable Databases." [Online]. Available: https://dev.maxmind.com/geoip/geoip2/geolite2/

[30] S. Carpendale, *Evaluating Information Visualizations*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 19–45. [Online]. Available: https://doi.org/10.1007/978-3-540-70956-5_2

[31] H. Lam *et al.*, "Empirical Studies in Information Visualization: Seven Scenarios," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 9, pp. 1520–1536, 2012.

[32] C. Lewis and J. Rieman, *Task-Centered User Interface Design*. University of Colorado, Boulder, Department of Computer Science, 1993.