BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

EVOLUTIONARY DESIGN AND OPTIMIZATION OF COMPONENTS USED IN HIGH-SPEED COMPUTER NETWORKS

EVOLUČNÍ NÁVRH A OPTIMALIZACE KOMPONENT POUŽÍVANÝCH VE VYSOKORYCHLOSTNÍCH POČÍTAČOVÝCH SÍTÍCH

EXTENDED ABSTRACT OF A PHD THESIS
ROZŠÍŘENÝ ABSTRAKT DISERTAČNÍ PRÁCE

AUTHOR — **Ing. DAVID GROCHOL**
AUTOR PRÁCE

SUPERVISOR — **prof. Ing. LUKÁŠ SEKANINA, Ph.D.**
ŠKOLITEL

BRNO 2019

Abstract

The research presented in this thesis is directed toward the evolutionary optimization of selected components of network applications intended for high-speed network monitoring systems. The research started with a study of current network monitoring systems. As an experimental platform, the Software Defined Monitoring (SDM) system was chosen. Because traffic processing is an important part of all monitoring systems, it was analyzed in greater detail. For detailed studies conducted in this thesis, two components were selected: the classifier of application protocols and the hash functions for network flow processing. The evolutionary computing techniques were surveyed with the aim to optimize not only the quality of processing but also the execution time of evolved components. The single-objective and multi-objective versions of evolutionary algorithms were considered and compared.

A new approach to the application protocol classifier design was proposed. Accurate and relaxed versions of the classifier were optimized by means of Cartesian Genetic Programming (CGP). A significant reduction in Field-Programmable Gate Array (FPGA) resources and latency was reported.

Specialized, highly optimized network hash functions were evolved by parallel Linear Genetic Programming (LGP). These hash functions provide better functionality (in terms of quality of hashing and execution time) than the state-of-the-art hash functions. Using multi-objective LGP, we even improved the hash functions evolved with the single-objective LGP. Parallel pipelined hash functions were implemented in an FPGA and evaluated for purposes of network flow hashing. A new reconfigurable hash function was developed as a combination of selected evolved hash functions. Very competitive general-purpose hash functions were also evolved by means of multi-objective LGP and evaluated using representative data sets. The multi-objective approach produced slightly better solutions than the single-objective approach. We confirmed that common LGP and CGP implementations can be used for automated design and optimization of selected components; however, it is important to properly handle the multi-objective nature of the problem and accelerate time-critical operations of GP.

Keywords

Evolutionary Algorithms, Cartesian Genetic Programming, Linear Genetic Programming, Network Monitoring, Network Application, Computer Network, Hash Function

Reference

GROCHOL, David. *Evolutionary design and optimization of components used in high-speed computer networks.* Brno, 2019. EXTENDED ABSTRACT OF A PHD THESIS. Brno University of Technology, Faculty of Information Technology. Supervisor prof. Ing. Lukáš Sekanina, Ph.D.

Contents

Chapter 1

Introduction

Many hardware providers have announced a support for 100 gigabit-per-second (Gbps) networks to overcome current 10–40 Gbps solutions [45, 57, 46]. The 400 Gbps and even 1 Tbps networks will be needed in a few next years, see Fig. 1.1. Commercial companies, data and supercomputer centers, and other entities around the world are now working towards launching 100 Gbps networks in order to benefit from faster communication and wider bandwidth in high-throughput requesting applications such as high-performance computing, high-quality video streaming or Internet of Things (IoT). Managing 100 Gbps networks requires more precise performance monitoring (involving bandwidth monitoring, traffic analytics and anomaly detection) than in the previous era.

In order to effectively monitor and analyze high-speed networks at the level of packet contents, *software defined monitoring* (SDM) concept has been developed [43]. Having less than 7 ns to process one packet in a 100 Gbps network, SDM performs the analysis using relatively simple (and so fast) hardware whose functionality (i.e. the rules of operation) are defined in the software. Unrecognized traffic is then processed by sophisticated algorithms in the software. The analysis is performed at the level of *network flows*, where a network flow is defined as a set of packets (with the same key features) that passed an observation point in the network during a given time interval.

Because there are only a few nanoseconds to process each packet, monitoring systems have to be carefully designed and optimized. Traffic monitoring systems perform many operations with flows (such as an extraction of the information from packet headers and a deep packet inspection to determine the application protocol) As these operations have to be executed for each packet in the flow, it is important to provide their efficient (highly optimized) implementations in high-speed networks.

Evolutionary algorithms (EAs) have successfully been used to design and optimize many applications. The automated search for a new or an improved piece of software is a typical task specifically for genetic programming (GP). GP can be used to improve existing software (e.g. [55]), create or optimize parallel programs (e.g. [56]) or automate generating full computer programs (e.g. [4]). In recent years, significant development and progress have been reported in evolutionary circuit design. In many cases these techniques were capable of delivering efficient circuit designs in terms of an on-chip area minimization (e.g. [89]), adaptation (e.g. [40]), fabrication variability compensation (e.g. [90]), and many other properties (see, for example, many requirements on synthetic benchmark circuits in [84]).

The main focus of this thesis is the optimization of low-level HW/SW components of network monitoring systems. It is important to identify the components that significantly influence performance in currently developed and future implementations of these systems.

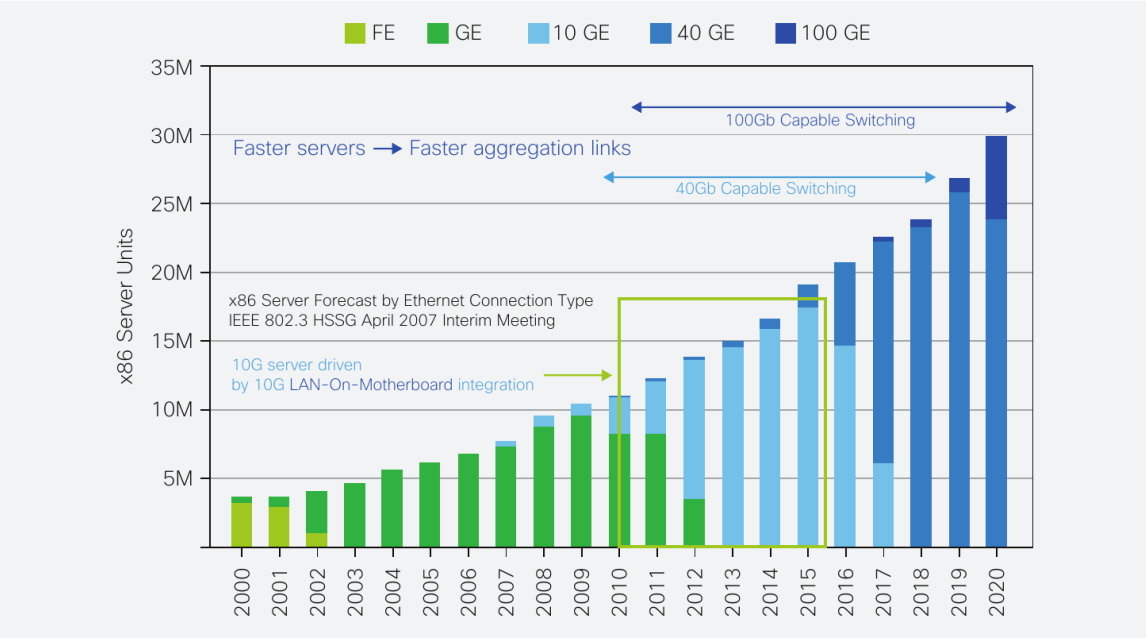

Figure 1.1: The network traffic is increasing rapidly in the last years (adopted from [11]).

The selected components require an optimization or a re-design not only in terms of functionality but also in terms of latency which is critical for high-speed networks. This thesis will explore how evolutionary algorithms (in particular genetic programming) can be employed to design and/or optimize selected components of high-speed network monitoring systems.

1.1 Research Objectives

We hypothesize that by using well-tuned evolutionary algorithms selected components of high-speed network monitoring systems can automatically be re-designed or optimized to improve their functionality and optimize other parameters (such as the implementation cost) in comparison with the state-of-the-art solutions.

The following research objectives were formulated:

1. To study network monitoring systems and identify their components suitable for automated optimization.
2. To define objectives and constrains that are important for an efficient optimization of the selected components.
3. To propose and implement single- and multi-objective variants of EAs including suitable fitness functions.
4. To validate the proposed approach and evolved solutions using relevant data sets.
5. To compare evolved solutions with the state-of-the-art implementations.

1.2 Abstract Outline

The extended abstract summarizes author's doctoral thesis. The thesis is composed as a collection of papers. The research contribution is presented in five peer-reviewed papers .

The extended abstract of the thesis is organized as follows. Chapter 2 surveys the state-of-the-art. It primarily includes the principles of network monitoring and evolutionary design of circuits and programs. A special attention is devoted to the classification of application protocols in hardware and to the hash function design because these two problems are later selected as the case studies for this thesis. The research methodology and overview of scientific papers constituting this thesis is given in Chapter 3. Finally, Chapter 4 concludes the thesis and suggests possibilities for future research.

Chapter 2

State of the Art

This chapter provides a necessary background needed to understand the research presented in this thesis. Chapter 2.1 introduces relevant concepts of computer networks and network applications that are important for monitoring and security of computer networks. The hash function design is presented in Chapter 2.3. Chapter 2.4 surveys the principles of genetic programming including the graph-based and linear-based variants of GP. Multi-objective approaches for evolutionary algorithms are presented in Chapter 2.5.

2.1 Computer Networks

A computer network is a telecommunication network that provides connections among end-systems in order to share resources.

Computer networks [70] consist of end-systems (such as personal computers, servers, mobile devices or IoT nodes) and network devices (such as routers, switches, bridges, firewalls). These devices are connected by different types of links (wired, wireless or optical). Computer networks can be local, connecting nodes in the boundary of space, or global, connecting nodes all around the world (the Internet). The basic architecture of a computer network includes:

- A technology for signal transmission,
- a technology for reliable data transmission and
- an application layer which provides services for users.

Several network architecture models have been proposed. The reference network architecture model is the ISO/OSI model [97], which contains seven layers, namely Physical, Datalink, Network, Transport, Session, Presentation and Application layers. The most widespread model is the TCP/IP model, which is based on the ISO/OSI model, but uses a simpler architecture than the ISO/OSI model. It has only four instead of seven layers used in the ISO/OSI model. A comparison of these models is shown in Table 2.1.

2.1.1 TCP/IP Model

The TCP/IP model [33, 85] has been designed with respect to reliability, independence of the transmission medium, decentralized and simple implementation. Table 2.1 shows all the layers and the typical protocols associated with each layer. Each layer uses services

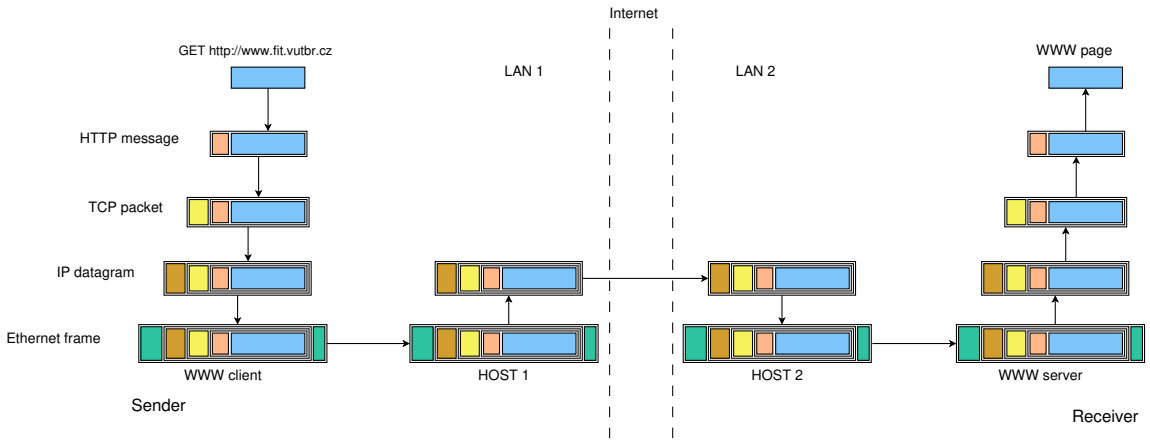

Figure 2.1: Example of the data encapsulation on the Sender side and the data decapsulation on the Receiver side.

provided by lower layers. Each layer adds a header or a footer to the data coming from the higher layers and delegates the encapsulated data to the lower layer for the next processing. Example of the data encapsulation on the sender side and decapsulation on the receiver side is shown in Figure 2.1. The lowest layer sends all the data with headers (footers) to a target device.

Network interface layer defines requirements on physical medium, electrical signals and optical signals. Examples of the network interface layer implementations are Ethernet, TokenRing, Frame Relay, FDDI or RS-232C. In addition to these requirements, the network interface layer defines the methods enabling the access to a physical medium.

Internet layer addresses and routes basic transfer structures (the so-called datagrams) containing the header and payload sections. Datagrams are routed using the best-effort delivery strategy, which tries to find a compromise between the shortest and the fastest paths through the entire network. If the datagram is lost during its transport via the network, the sender has to arrange for its re-sending. The internet layer typically uses five protocols. The Internet Protocol (IP) provides services for transport layer protocols. It distinguishes devices in the network. Two versions of IP are currently used IPv4 and IPv6, see Figure 2.2. Next protocols are Address Resolution Protocol (ARP) and Reverse ARP (RARP). The ARP translates an IP address to a MAC address and RARP translates a MAC address to an IP address. Internet Group Message Protocol (IGMP) is used for logging to multicast groups. Internet Control Message Protocol (ICMP) sends error messages in the networks, for example, a device goes offline or a service is unavailable. All internet layer protocols have to be implemented in the operation system.

Transport layer ensures logical connections between processes on the end devices connected through the IP. The logical connection is identified by a port number. The transport protocol divides the data coming from the application layer to smaller pieces and adds a header in order to form a packet. The packets create a sequence which is called the *network flow*. The network flow is defined as a set of packets with the same key features and is passed by an observation point in the network during a given time interval. The transport layer uses two protocols. (i) The Transmission Control Protocol (TCP) ensures reliable connections. It means that all the data sent from a sender will be delivered to a recipient correctly. The TCP adds special packets to the communication to establish the connection, finish the connection, confirm an acceptance, synchronize all entities, etc. (ii)

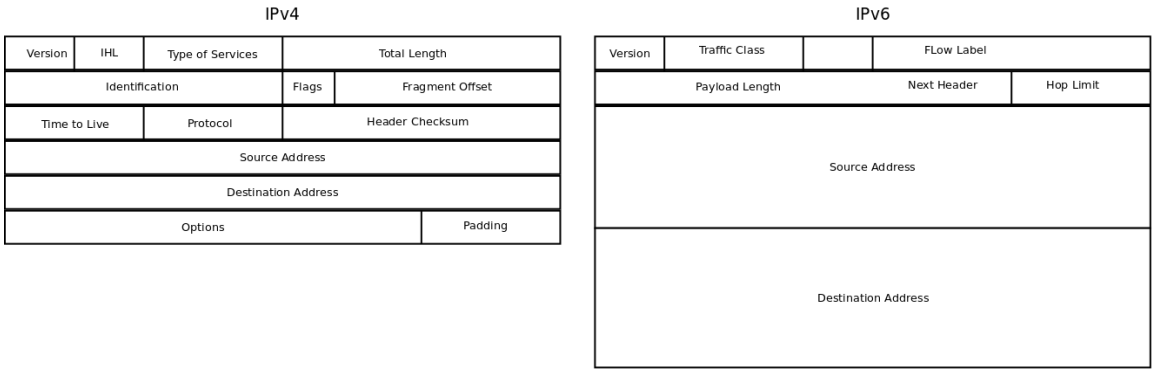

Figure 2.2: Comparison of IPv4 header (left) and IPv6 header (right).

Table 2.1: Comparison of ISO/OSI model and TCP/IP model.

ISO/OSI model	TCP/IP model	Group of Layers	Examples
Application Layer Presentation Layer Session Layer	Application Layer	Application Layer	Web Pages and Internet browsers
Transport Layer Network Layer	Transport Layer Network Layer	Internetwork Layer	TCP/IP Software
Data Link Layer Physical Layer	Data Link Layer Physical Layer	Hardware Layer	Ethernet ports, cables and ethernet drivers

The User Datagram Protocol (UDP), in fact, creates unreliable connections as there is no mechanism to check if a packet is delivered correctly. The data transfer is faster using UDP than TCP because there are no additional control packets. UDP is employed by services which need a fast transmission but can tolerate some undelivered packets, for example, Video on Demand (VOD), audio stream, etc.

Application layer covers many different application protocols developed for specific applications. Each protocol uses either TCP or UDP as transport protocol. Well-known protocols such as HTTP, SMTP, SIP, SSH have a port number assigned. The port numbers are divided into three ranges. The well-known ports, also known as the system ports, belong to the interval from 0 to 1023. These applications have to e strictly registered. The second group of ports (with numbers from 1024 to 49151) is a subject to registration at IANA organization. For the last group, with the so-called dynamic (or private) ports in the range from 49152 to 65535, there are no specific requirements for registration.

2.2 Network Monitoring

The network monitoring is crucial for ensuring the correct functionality of computer networks. It is based on probing of device states, traffic analysis and collecting traffic information. Results of monitoring are useful for administrators to improve network security, performance and functionality. Two types of network monitoring techniques exist [78, 66, 54]. (i) The *active monitoring* lies in injecting test traffic into the network and analyzing its impact. These tests can reveal real-time problems such as packet loss, jitter, insufficient bandwidth, unknown device status, latency and measure the quality of services (QoS). The

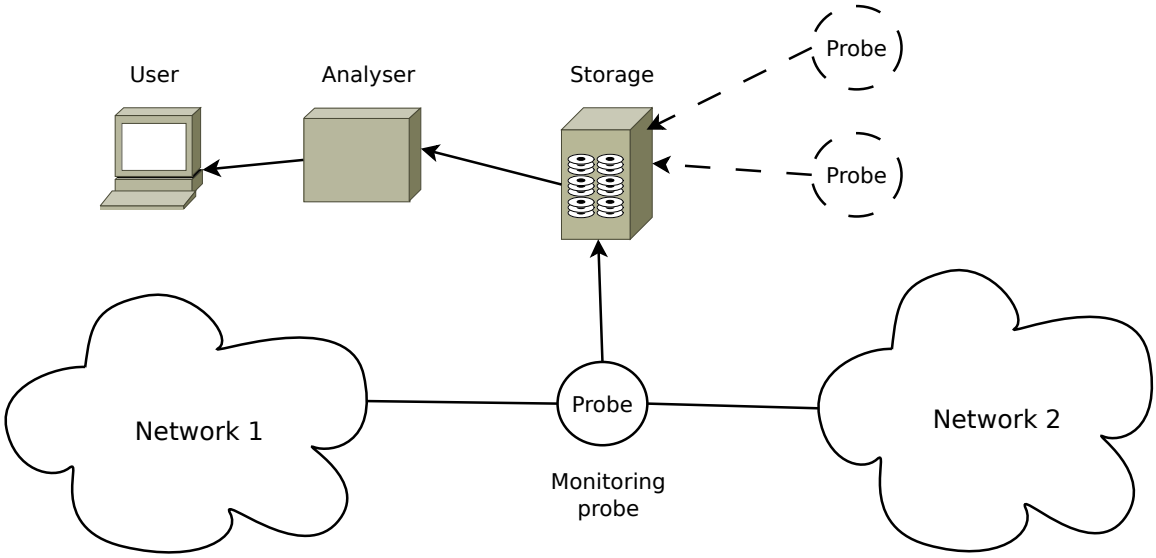

Figure 2.3: Schema of a monitoring system architecture.

active monitoring can thus relatively quickly detect network problems. The disadvantages of the active monitoring are increased network traffic and missing information about the data in the packed payload because only injected packets are analyzed. (ii) The *passive monitoring* is based on an analysis of real network traffic without injecting any new network traffic. It requires either special network devices to capture the network data or a built-in support in switches and other network devices. The passive monitoring provides more opportunities for the analysis than active monitoring. For example, the volume of the data generated by a device or anomalies in the traffic behavior can be detected. Any specific application, any user or any specific traffic may be observed and analyzed. The long-term statistics created from captured data are of a great importance for future infrastructure planning and upgrades. A monitoring probe is typically inserted between two networks, see Figure 2.3. The network probes typically send the data to a collector that analyses the data and distribute them to users or makes certain automatic or semi-automatic actions such as blocking of specific devices (users), applications or attacks in real-time.

Current high-speed networks operate at 10 Gbps and 40 Gbps. Solutions for 100 Gbps networks have been already demonstrated [43]. The specifications for 400 Gbps and even 1 Tbps are under design. High-speed networks require a novel approach to traffic monitoring because the monitoring systems used in 10 Gbps networks do not have enough throughput. A common feature of software implementations of network monitoring systems is that the monitoring probe is flexible and easily adapts to a given network. However, the software solution is often insufficient in terms of performance. Hardware implementations of monitoring systems, based on field-programmable gate arrays (FPGA) or application-specific integrated circuits (ASIC), usually have a sufficient throughput but it is often difficult to adapt them to a certain network or add new monitoring features.

A new approach to high-speed network monitoring known as Software Defined Monitoring (SDM) has recently been developed [43, 45]. The SDM combines hardware and software approaches. The SDM consists of three fundamental parts: i) a special hardware card with an FPGA based network monitoring accelerator, ii) a firmware controlling the data preprocessing in hardware and iii) user applications, see Fig. 2.4. The main benefit

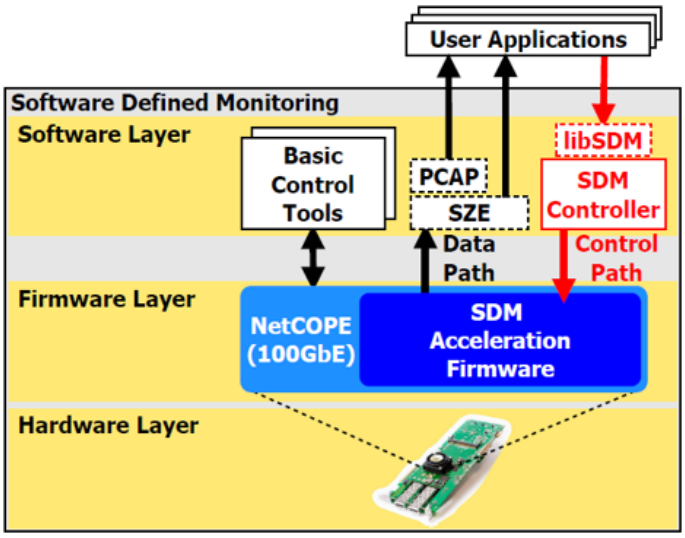

Figure 2.4: Software Defined Monitoring system architecture (adopted from [43]).

of SDM is its ability to control the hardware preprocessing of network flows by means of software applications. The control software sets the rules for the hardware accelerator with respect to the particular flows, groups of flows, devices (users) or selected protocols. It also sets various actions specifying how to discard packets, collect basic network characteristics, capture all packets for detailed analysis, etc. A rule or a set of rules is defined to preprocess each flow. Because the software part has to process only a small portion of the traffic (already preprocessed and aggregated data) it can be executed on a standard multi-core processor. About 80% of flows can be processed in hardware after the learning phase of the SDM system is finished [43]. However, during the learning phase, the software has to handle most of the flows.

2.2.1 Processing of Network Traffic and Network Flows

This section briefly describes the flow processing in the SDM system and identifies the most time-critical operations.

Network Traffic Processing

In the current networks, which are mostly based on TCP/IP model [72], a packet is formed by headers and payload. Every layer has its own header for identification and processing. A flow can be uniquely identified by a 5-tuple extracted from these headers: source and destination internet protocol addresses (IPv4 or IPv6), source and destination ports and transport protocol (mostly TCP or UDP). Each flow represents one direction of the communication between two applications on the devices. In the flow-based monitoring, we deal with the basic flow characteristics such as the flow length (the number of packets or bytes) and timing (the start time, the end time, the duration). These characteristics are often amended by certain interesting information from an application layer such as the type of protocol or some information extracted from the payload.

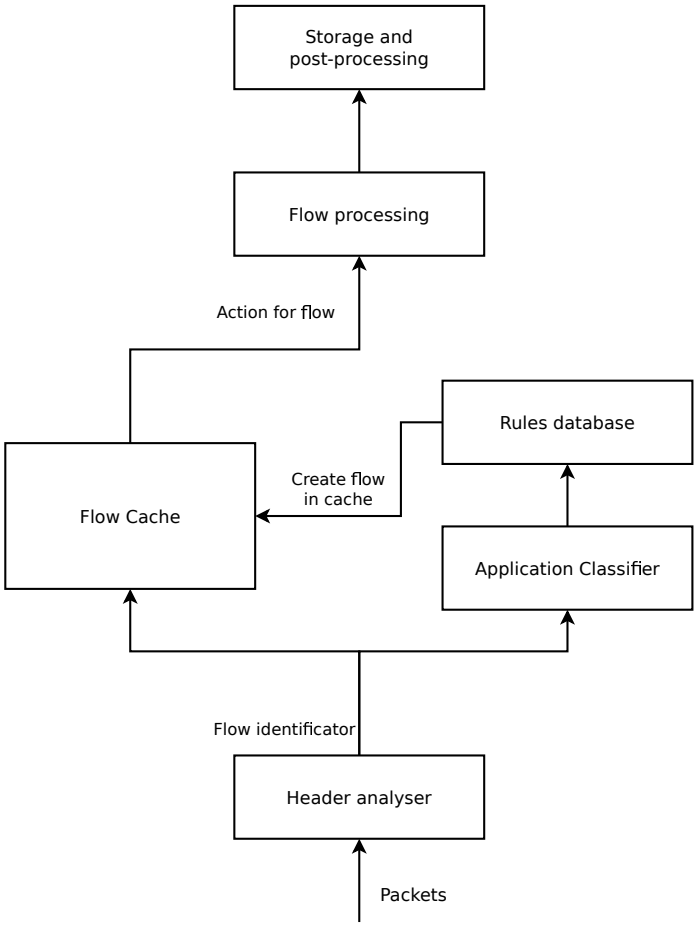

Figure 2.5: Packet processing in monitoring systems such as SDM.

Flow Processing

Fig. 2.5 shows a typical packet processing pipeline in the monitoring systems such as SDM. At first, the 5-tuple identification of the packet is extracted from packet headers. The packet is assigned to a flow record if the corresponding flow already exists in the flow cache. Otherwise, a new flow record is created in the flow cache and rules for the processing of the packets belonging to this flow are defined. There are different types of rules, for example, "capture all traffic", "get basic characteristics" or "get advanced characteristics". These rules can affect complete network traffic, a specific user, a subnet, a specific protocol or a group of protocols (e.g. communication protocols).

One of the challenges in network monitoring is the identification of application protocols. The research in the area of application identification has come up with distinct approaches to identify the applications carried out in the traffic. These approaches differ in the level of detail that is utilized in the identification method. The most abstract one is the *behavioral analysis* [38, 95]. Its idea is to observe only the port numbers and destination of the connections per each host and then to deduce the application running on the host by its typical connection signature. If more details per connection are available, the *statistical fingerprinting* [65] comes into play. In this case, a feature set is collected per each flow and the assumption is that the values of the feature set vary across applications, and hence, the applications leave a unique fingerprint. Behavioral and statistical fingerprinting

Table 2.2: Characteristics of different speed links. *Packet size is 64 bytes **CPU frequency = 3.6GHz

Link speed [Gbps]	Packets* per second	Time to process one packet [ns]	approximate CPU** clocks cycles
1	1 953 125	512.0	1843
10	19 531 250	51.2	184
40	78 125 000	12.8	46
100	195 312 500	5.12	18
400	781 250 000	1.28	5

generally classifies the traffic into the application classes rather than into the particular applications. The reason is that different applications performing the same task often exhibit similar behavior. For instance, application protocols such as Oscar (ICQ), MSN and XMPP (Jabber) transport interactive chat communications, and hence, they exhibit a similar behavior, which is very hard to differentiate for the monitoring system. The inability to distinguish applications within the same class is in some situations seen as a drawback, for example, when it is necessary to block/capture an application while other applications of the same class have to remain running. The approach utilizing the greatest level of detail is called *deep packet inspection*. It identifies applications based on the packet payload. The payload is matched with known patterns (defined, for example, by regular expressions) derived for each application [80].

The L7 filter [25] is a popular program for the application protocol identification, which utilizes regular expressions to describe the application protocols. It performs pattern matching in network flows. If a known pattern is matched in the payload, the corresponding application protocol is assigned to the network flow. Current processors are not powerful enough to achieve 100 Gbps throughput for the regular expression matching. The throughput of L7 decoder is less than 1 Gbps per one CPU core even for the latest Xeon processors [31, 32]. In order to achieve 100 Gbps throughput, it is necessary to use highly optimized hardware accelerators.

2.2.2 FPGA Based Accelerators

Table 2.2 shows how requirements on CPUs are growing with increased link speed. As a processor-based network monitoring is only applicable to 10 Gbps, hardware acceleration is needed for faster links.

Paxson et al. [69] argue that these performance requirements should be met by leveraging a high degree of possible parallelism that is inherent to network traffic monitoring. FPGAs, as well as ASICs, may deliver a vast support of parallelism. However, only FPGAs enable possibility to prototype and implement critical components for various network applications at the highest speeds while the optimized ASICs follow broad deployment a few years later. FPGAs are extensively used in the so-called hardware-accelerated network cards to implement the first line of network traffic processing such as monitoring, forwarding and other applications [1, 44].

FPGAs consist of programmable routing network and basic building blocks such as look-up tables (LUTs), registers, and block memories. A particular setup of the routing network defines the interconnection of these components. The LUTs serve to implement combinational logic while registers and block memories serve to keep the stateful information.

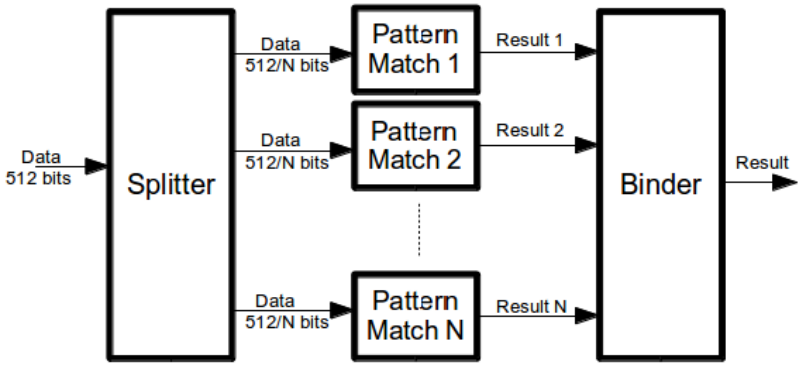

Figure 2.6: Increasing the throughput by multiple pattern matching units.

Modern FPGAs contain millions of LUTs and registers and thousands of block memories with the total capacity of hundreds MB [94]. All these components may, theoretical, work in parallel, independently of each other, and provide enormous computation power with low energy consumption in tens of Watts per chip. Moreover, FPGAs targeting the network market provide more than a hundred of high-speed transceivers allowing for connection to high-speed network links (e.g. high-end Virtex UltraScale+ FPGA offers up to 4 Tbps of aggregated transceiver throughput [93]).

The crucial task is to transform a high-level description of the circuit (for example, written in VHDL or SystemC) into an effective implementation in FPGA from the perspective of meeting the timing and resource constraints.

In recent years, many researchers have proposed high-speed pattern matching hardware architectures, which utilize the fine-grained parallelism of FPGA technology. Mapping of the regular expressions matching to an FPGA was first explored by Floyd and Ullman [26], who showed that pattern matching realized by a Nondeterministic Finite Automaton (NFA) can be implemented using a programmable logic array. Sindhu et al. [82] proposed an efficient mapping of NFAs to FPGA and Clark et al. improved the implementation by a shared decoder [12, 13] which significantly reduced the amount of consumed logic resources. The AMTH (At Most Two-Hot encoding) architecture [96] provides another improvement of the NFA implementation in the FPGA. The combination of one-hot and binary encoding reduces the amount of flip-flops, representing the NFA states.

Several authors introduced an optimized mapping of Perl Compatible Regular Expressions (PCRE), which are widely used in Intrusion Detection Systems (IDS), to the FPGA. Sourdis et al. published in [83] an architecture that allows for the sharing of character classes, static sub patterns and introduced components for efficient mapping of constrained repetitions to the FPGA. Lin et al. created an architecture for sharing infixes and suffixes [58]. Nevertheless, these optimizations are relevant only for large sets of PCREs in IDS systems.

The throughput of a pattern matching circuit is determined by the number of bytes processed within one clock cycle and frequency of the hardware matching unit. The FPGA technology limits the maximum frequency to several hundreds of MHz. To increase the processing speed, the NFA can be modified to process multiple bytes per one clock cycle [8]. Unfortunately, with the increasing size of the NFA input, the amount of NFA transitions grows exponentially. As a result, the hardware matching unit consumes more FPGA resources and its frequency decreases rapidly.

The throughput can be increased by introducing multiple parallel matching units. These units need additional logic resources and buffers to distribute the network data to the matching units and join the results. The overhead of parallel processing is illustrated in Fig. 2.6. First, the splitter has to assign the sequence number for every packet and store the packet to a buffer. The packet data are then sent with a lower rate to one of the parallel matching units. The units perform pattern matching and send the results to a binder, which contains buffers to put the results in the right sequence order.

Introducing the parallel matching units can improve the matching speed up to 100 Gbps, but only at the cost of significant overhead in terms of latency, FPGA logic resources and memory buffers. This overhead is avoided by focusing on highly optimized hardware architectures with high throughput and low latency [59, 60].

2.3 Hash Function Design

Hash functions are often employed in hardware accelerators of network monitoring systems. They are responsible for searching in the rule table, for distributing data to process units and for storing the flows data to database. For example, in the distribution unit, a hash function is called for each packet. In order to maximize the performance of network monitoring systems, hashing has to be not only of a high quality, but also fast.

A *hash function* is a mathematical function h that maps an input binary string (of length l_D) to a binary string of fixed length (l_R), $h : D \to R$, where $l_D > l_R$. The output value is called a hash value or simply hash [50]. The definition of hash function implies the existence of *collisions*, i.e. $h(d) = h(d')$, where $d, d' \in D$ are two different input messages. An important requirement imposed on hash functions is that a small change in the input should generate a large change in the output, which is called the *avalanche effect*. Good hash functions usually satisfy both criteria – maximizing the avalanche effect and minimizing the collision rate.

Two major types of hash functions exist, cryptographic and non-cryptographic. The cryptographic hash functions are suitable for cryptographic applications [86]. They have to satisfy many requirements, e.g.:

- *practical efficiency* – for $d \in D$ it is computationally efficient to find a hash value $r \in R$ s.t. $h(d) = r$;
- *first preimage resistance* (one-way) – for $r \in R$ it is computationally infeasible to find an input value $d \in D$ s.t. $h(d) = r$;
- *second preimage resistance* (weak collision resistance) – for $d \in D$ it is computationally infeasible to find a value $d' \in D$, s.t. $d' \neq d$ and $h(d') = h(d)$;
- *collision resistance* (strong collision resistance) – it is computationally infeasible to find two distinct values $d', d \in D$, s.t. $h(d') = h(d)$.

These requirements lead to more complicated construction of hash functions and, hence, the cryptographic hash functions need more time to compute the hash value than the non-cryptographic hash functions. The cryptographic hash functions have many applications, for example, in message authentication tools, digital signatures or in other forms of authentication.

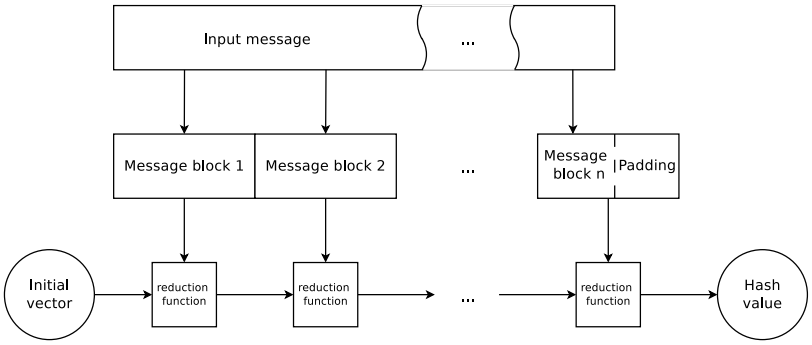

Figure 2.7: Merkle–Damgård construction of hash functions.

The non-cryptographic hash functions have to satisfy weaker requirements, but the practical efficiency and collision resistance are also important. These properties are often used to quantify the hashing quality of non-cryptographic hash functions.

Because the input size is usually arbitrary, hash functions are often designed using a pipelined (Merkle–Damgård) construction, see Fig. 2.7. It means that an input message is divided into blocks of a fixed size and processed block by block. The block is processed one at a time with an inside reduction function, each time combining the input block with the output of the previous round. The size of the output is typically the same as the size of the hash value. The last round produces the hash value. The last block of the message is typically padded with zeros to the required size.

The non-cryptographic hash functions have many applications, for example, in hash tables, search duplication, caches, bloom filters [61, 71, 36]. The *hash table* is a data structure used to implement an associative array, a structure which maps keys to values, see Figure 2.8. Hash tables have many applications, such as database indexing, object representation in programming languages or sets. Because hash functions produce collisions, it is necessary to resolve them in the hash tables. A well-known technique is *separate chaining*, where each slot in the hash table refers to a linear list that contains the records having the same hash. While determining the slot for a given input is performed in constant time, a particular record have to be searched sequentially. Next approach the collision

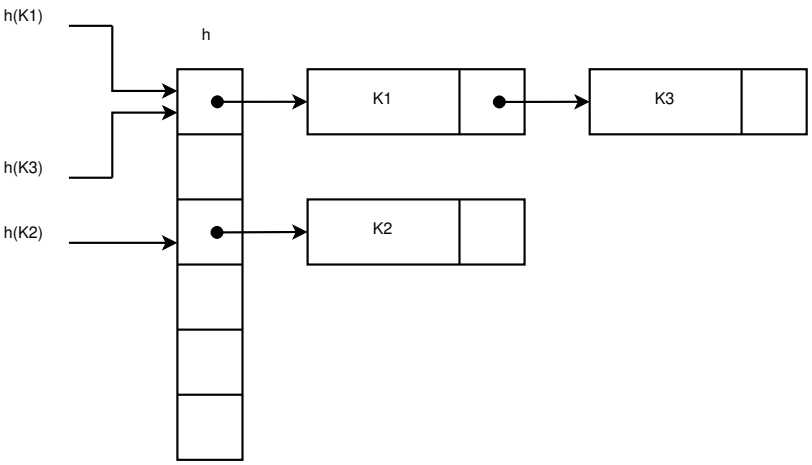

Figure 2.8: Example of hash table with size 6 slots, utilizing the separate chaining.

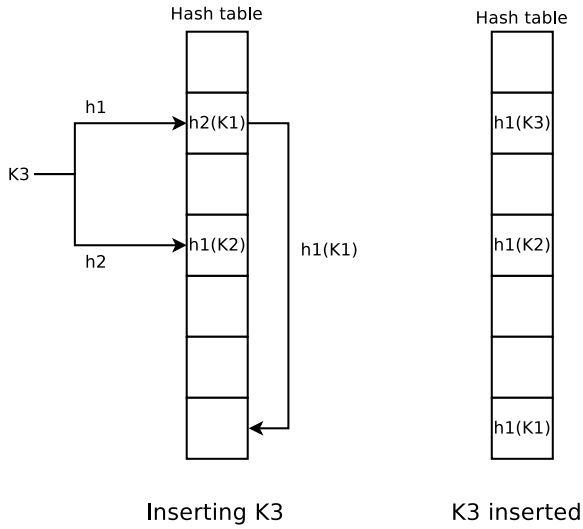

Figure 2.9: Example of inserting key to hash table with Cuckoo hashing.

resolving is the *open addressing*. This method searches (using some algorithms) for an alternative position in the hash table, where to store the data. Another approach, *cuckoo hashing* [68], uses two hash functions. A key is hashed by both hash functions and the data are stored to an empty slot indexed by one of them. If both slots are occupied, one of the keys stored in the table is rehashed by the other hash function and stored there, see example in Figure 2.9.

Many (non-cryptographic) hash functions have been proposed, for example, DJBHash [6], DEKHash [50], FVN (Fowler-Noll-Vo) [27], One At Time and Lookup3 [36]. MurmurHash2 and MurmurHash3, which are utilized in many open source projects, are hash functions suitable for general hash-based lookup [2]. CityHash is a family of non-cryptographic hash functions designed for fast hashing of strings [71]. Additional details are available in Paper II.

In addition to the general purpose non-cryptographic hash functions, there are also exist application-specific hash functions. They address specific properties of a particular application and, therefore, can be better (with respect to these properties) than the general-purpose hash functions. For hashing of network flows, the so-called XOR folding has been proposed [9]. Its implementation works with inputs of fixed size and is optimized in terms of performance.

SHMHasher [3] is a framework developed for evaluation of hash functions. It provides a test suite to evaluate the distribution, collision and performance properties of non-cryptographic hash functions. It contains many hash functions that can be used for a comparison. We used this framework in Paper V to measure the performance of hash functions.

2.4 Evolutionary Design

Evolutionary algorithms (EAs) [75] are inspired by the principles of biological evolution which is seen as an excellent optimization system. EAs are a class of stochastic optimization algorithms in which a population (a set) of candidate solutions is modified by genetic operations in order to solve a particular optimization problem. The quality of candidate

solutions is evaluated by means of the fitness function. A general evolutionary algorithm works as follows:

1. Initialize the population of candidate solutions (individuals).
2. Evaluate all individuals to determine their *fitness value*.
3. If termination conditions are met then stop. The result of EA is the individual with the best fitness value.
4. By means of a selection method select individuals from the population to a set of parents.
5. Create a set of offspring by applying genetic operators on the parents:
   (a) *Reproduction* – copy an individual to the offspring set unchanged
   (b) *Recombination* – exchange some parts of two or more individuals
   (c) *Mutation* – randomly modify some parts of an individual
6. Create a new population using the set of parents and offspring
7. Continue with step 2.

Many variants of evolution algorithms have been proposed in the literature, for example, evolution strategy [74], differential evolution [73], genetic algorithm [15] and genetic programming [52].

Genetic programming (GP) [52, 53] is primarily used for automated design of computer programs. Candidate programs are represented in memory as syntactic trees in the so-called tree version of GP. Nodes of the tree represent operations (arithmetic, logic, control etc.) and leaves contain terminal symbols such as program's inputs or constant values. During evolution, every candidate program is executed on a training data set in order to obtain its fitness value. Genetic operators randomly modify one candidate tree (mutation) or two or more candidate trees (swapping of subtrees) in crossover. The resulting tree is evaluated using a test set to validate its behavior on unseen data.

Other variants of GP use a different encoding of candidate programs. Cartesian GP and Linear GP are described in greater detail in the next chapter because they are relevant for this thesis.

2.4.1 Cartesian Genetic Programming

Cartesian genetic programming (CGP) has been developed by Miller since 1999 [64] and has been utilized in many applications as summarized in monograph [62]. A typical application of CGP is evolutionary circuit design. The idea of evolvable hardware and automated circuit design by means of artificial evolution was introduced by Higuchi et al. in 1993 [34]. A recent survey of the field covering key subfields (evolutionary hardware design and adaptive hardware) is available in [79]. In CGP, a candidate solution is modeled as a directed acyclic graph and represented in a 2D array of $n_c \times n_r$ processing nodes. Each node can perform one of the n_a-input functions specified in Γ set. The setting of n_c, n_r and Γ significantly influences the performance of CGP [63, 29].

The remaining parameters of CGP are the number of primary inputs (n_i), the number of primary outputs (n_o), and the level-back parameter (L) specifying which columns can

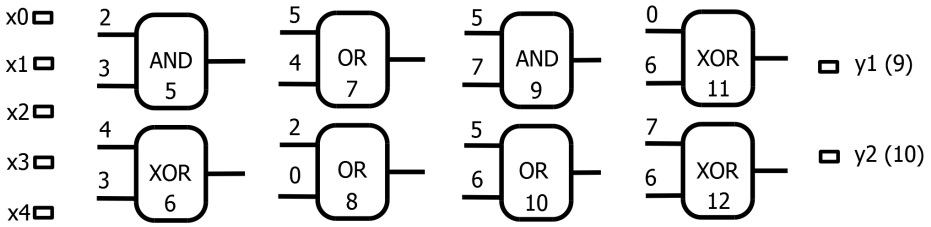

Figure 2.10: Example of a combinational circuit in CGP with parameters: $n_a = 2$, $n_i = 5$, $n_o = 2$, $L = 4$, $n_c = 4$, $n_r = 2$, $\Gamma = \{\text{AND } (0), \text{OR } (1), \text{XOR } (2)\}$. Gates 8, 11 and 12 are not utilized. Chromosome: 2,3,0; 4,3,2; 5,4,1; 2,0,1; 5,7,0; 5,6,1; 0,6,2; 7,6,2; 9, 10. The last two integers indicate the outputs of the circuit.

be used as inputs for a given gate. The primary inputs are labeled $0 \ldots n_i - 1$. The outputs of all nodes are labeled $n_i - 1 \ldots n_c \cdot n_r + n_i - 1$ and considered as addresses where the connections can be fed to. In the chromosome, each n_a-input node is then encoded using $n_a + 1$ integers (n_a inputs and a node function). Finally, for each primary output, the chromosome contains one integer specifying the connection address. In CGP, the encoding is redundant because some nodes, some of their inputs or some primary inputs need not be used in the phenotype.

Algorithm 1: CGP

Input: CGP parameters, fitness function, original circuit p
Output: The highest scored individual and its fitness

```
1 P ← CreateInitialPopulation(p);
2 EvaluatePopulation(P);
3 while ⟨terminating condition not satisfied⟩ do
4 |   α ← SelectHighest-scored-individual(P);
5 |   if fitness(α) ≥ fitness(p) then
6 |   |   p ← α;
7 |   P ← {p} ∪ {λ offspring of p created by mutation};
8 |   EvaluatePopulation(P);
9 return p, fitness(p);
```

CGP utilizes a search method known as $1 + \lambda$, where λ is the population size [62]. The initial population is randomly generated or seeded using conventional solutions. A new population consisting of λ individuals is generated by applying the mutation operator on the best individual of the previous population. The mutation operator randomly modifies h integers of the chromosome. The evolution is terminated after producing a given number of generations or a suitable solution is discovered, see Algorithm 1.

In the standard CGP used for combinational circuit evolution, the number of primary inputs n_i and outputs n_o is set accordingly to the requirements of the target circuit and Γ contains a set of Boolean functions. Figure 2.10 shows an example of a circuit and a corresponding chromosome.

A candidate circuit is evaluated by checking its responses for all possible input combinations. In order to accelerate the fitness function evaluation on a common processor, a bit-level parallel simulation of a candidate combinational circuit is employed. Contrasted

```
double LGP (double x ){
  r[0] = x

  r[2] = r[0] * r[0]
  r[1] = r[2] + r[0]
  r[3] = r[1] + r[0]
  r[1] = r[1] + r[4]
  r[0] = r[1]
  return r[0]
}
```

Figure 2.11: Example of a candidate program in LGP.

to a naïve simulation, in which 2^{n_i} vectors are sequentially submitted for evaluation (where n_i is the number of primary inputs), the bit-level parallel simulation exploits the fact that current processors enable performing bitwise operations over two w-bit operands in parallel [62]. Hence, the input vectors are grouped into w-bit words and simulated in parallel. The obtained speedup is w on a w-bit processor, for example, 64 × on a 64 bit common personal computer.

Although various new designs have been discovered using the standard CGP, the method is not directly applicable for the design of large combinational circuits because the fitness evaluation time grows exponentially with the number of primary inputs. Moreover, the number of requested fitness evaluations can easily go into millions, even for small (but nontrivial) circuits such as 4 bit multipliers. This problem has partially been eliminated by introducing circuit decomposition techniques at the representation level [87, 81] and formal verification methods in the fitness function [89]. Other successful applications of CGP have been proposed in domains in which candidate circuits are not evaluated using all possible input combinations (see, e.g., hash functions [41], image operators [88] or classifiers [40]).

The modern FPGAs contain 4- or 6-inputs LUTs. There are only a few papers dealing with the evolutionary circuit design at the level of 4-input LUTs [10, 41] and no paper dealing with 6-input LUTs. Unfortunately, the bit-level parallel simulation is inefficient for circuits consisting of LUTs because their logic function has to be emulated using a sequence of binary logic operations. As discussed in Paper I, employing CGP with 6-input LUTs (each of them encoded using 64 bits in the chromosome) would lead to long chromosomes, complex search spaces and very inefficient search procedures.

2.4.2 Linear Genetic Programming

Linear genetic programming (LGP) [7, 67, 92] is a variant of GP which uses a linear representation of candidate programs. Every program is composed of operations (called instructions) that are executed in a register machine. Operands, intermediate results and final results are stored in registers or in an external memory. Example of a candidate program is given in Figure 2.11. Linear GP evolves sequences of instructions in a machine language.

An instruction is typically represented by the instruction code, destination register and two source registers, for example, $[+, r0, r1, r2]$ represent the operation $r0 = r1 + r2$. The program result is returned in a predefined register. The number of instructions in a candidate program varies during the evolution, but the minimal and maximal size are defined. The number of registers available in the register machine is constant. The function

set known from GP corresponds with the set of available instructions. The instructions are general-purpose (e.g., addition and multiplication) or domain-specific (e.g., read sensor 1). Conditional and branch instructions are important for solving many problems. As in tother branches of GP, protected versions of some instructions (e.g., a division returning a value even if the divisor is zero) are employed in order to execute all programs without exceptions (such as division by zero).

LGP is usually used with a tournament selection, one-point or two-point crossover and a mutation operator modifying either the instruction type or the register index. Advanced genetic operators have been proposed for LGP, for example [21, 22].

Like in other GP branches, the most computationally expensive part of LGP is the fitness function evaluation. In order to obtain the program's fitness score, the candidate program is executed on a set of training inputs, its outputs are collected and compared with desired values.

An individual can contain unused code parts, called *introns*, which do not affect the fitness value. However, the introns slow down the program execution. If introns are detected and eliminated, the evaluation time can be significantly reduced. According to [7], the existence of introns is important for the evolution process. Introns may act as a protection that reduces the effect of the variation process on the effective code.

The fitness function is typically focused on functionality, but other parameters of candidate programs can be optimized, such as the number of used instructions, execution time or power consumption of the processor.

2.4.3 Evolution of Hash Functions

Hash functions were successfully designed by evolutionary algorithms in recent years. The main advantage of EAs is that they are capable of producing high-quality hash functions optimized for a given application domain. Hash functions were evolved with genetic algorithms [76], tree GP [24], grammatical evolution [5] and Cartesian GP [91]. Both scenarios – application-specific hash functions (see, e.g., [42, 47, 51]) and general-purpose hash functions (see, e.g., [24, 39]) – were addressed in the literature. Relevant details are given in papers II, III, IV and V.

The fitness functions used in EAs developed for hash function design have been mostly focused only on the quality of hashing, usually expressed in terms of the collisions resistance, avalanche effect and distribution of outputs. The execution time of hash functions were not addressed by the GP literature before my research has been initiated.

2.5 Multi-Objective EAs

Previous chapters have dealt with single-objective EAs that produce solutions with respect to only one objective [18]. In many real-world problems such as the hash function design problem discussed in chapter 2.4.3, there are two or more optimization objectives that are conflicting. A simple approach is to combine several objectives into one (scaled) fitness function. Modern EAs, however, provide many useful techniques for truly multi-objective optimization [23, 16, 49].

A general multi-objective optimization problem is defined as follows:

Table 2.3: Solution relations in a multi-objective approach [23].

relation	notation	interpretation
strictly dominates	$x \prec\prec y$	$f_n(x) > f_n(y) \forall n$
dominates	$x \prec y$	$f_n(x) \geq f_n(y) \forall n \wedge \exists i : f_i(x) > f_i(y)$
weakly dominates	$x \preceq y$	$f_n(x) \geq f_n(y) \forall n$
incomparable	$x \parallel y$	$\neg(x \preceq y) \wedge \neg(y \preceq x)$
indifferent	$x \sim y$	$f_n(x) = f_n(y) \forall n$

$$\begin{gathered} \text{minimize/maximize } F(x) = (f_1(x), f_2(x), f_3(x), ...f_n(x)) \\ \text{subject to } g_i(x) \geq 0, i = 1, ..., m, \\ h_j(x) = 0, j = 1, ..., p, \end{gathered} \tag{2.1}$$

where f_i is the objective (fitness) function, n is the number of objectives and x is an individual. g_i and h_i are inequity and equity constraints, where m and p is the number of constrains. Various multi-objective algorithms have been proposed. These algorithms use different approaches to combine the optimization criteria and select new candidate solutions. A straightforward approach is to assign a weight for each fitness function. The final fitness function is the sum of the weighted fitness values:

$$fitness = \sum_{i=1}^{n} w_i f_i(x), \tag{2.2}$$

where w_i is the weight for i-th fitness function. Another approach is based on lexicographical sorting, in which individuals are gradually sorted by fitness values according to user preference. For example, VEGA algorithm [77] randomly divides the population into n subsets, where n is the number of objectives. Each subset is evaluated by one fitness function. The new population is formed by individuals from all subsets selected by a selection algorithm based on the fitness value.

The most successful multi-objective algorithms are based on the principle of *Pareto dominance*. We say that solution x Pareto dominates solution y if the following two conditions are fulfilled:

1. Solution x is better than solution y in at least one objective.
2. Solution x is no worse than solution y in all objectives ($x \prec y$).

This is formally captured by relation (the objective is to maximize F_i):

$$x \prec y : \forall n f_n(x) \geq f_n(y) \wedge \exists i : f_i(x) > f_i(y) \tag{2.3}$$

Table 2.3 summarizes all important relations between two solutions.

The set of solutions (out of all solutions) that are not dominated by any other solution forms *Pareto-optimal front* or *Pareto-optimal set*. Pareto front can also be constructed using solutions from a given population, i.e. using a subset of all possible solutions. Fig. 2.12 shows Pareto front (black) containing solutions which dominate the other solutions (white) in the population. The ultimate goal of the multi-objective algorithm is to find the Pareto-optimal set of solutions.

Evaluation algorithms utilizing the Pareto dominance employ different strategies to select individuals to the offspring population. Their main purpose is to maintain the diversity

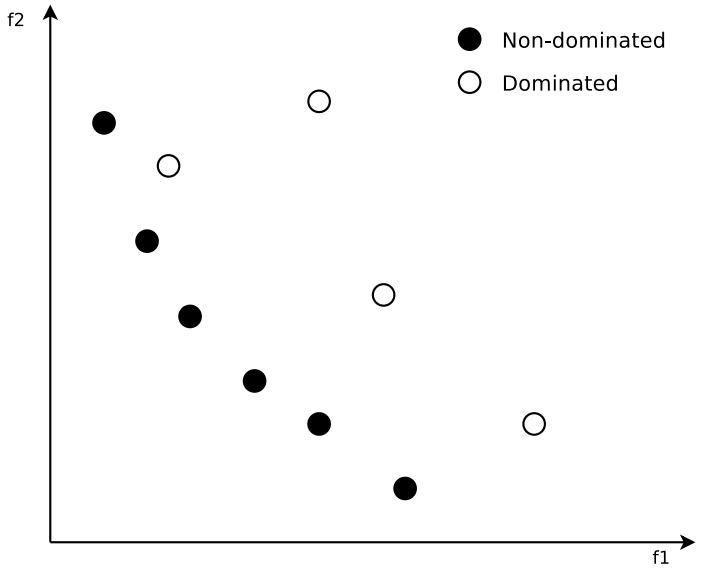

Figure 2.12: Individuals on the Pareto front (black points) dominate the remaining individuals (white points) in the population. f_1 and f_2 have to be minimized.

of the population. Selected multi-objective algorithms are described in detail in the next paragraphs.

Strength Pareto Evolutionary Algorithm 2

The Strength Pareto Evolutionary Algorithm 2 (SPEA2) was proposed by Zitzler et al. in 2001 [98, 48]. SPEA2 uses two sets of individuals (population and archive). The archive includes all non-dominated individuals from populations. If the archive is oversized, the number of individuals in the archive is reduced by the *truncation operation*; otherwise, if the archive is undersized, it is filled by the best dominated individual(s) from the population. The truncation operation performs the nearest neighbor algorithm on the individuals included in the archive. The individual with a minimal distance to another individual is chosen to be removed from the archive. The truncation operation removes individuals from the archive until the required size of the archive is reached.

SPEA2 uses a binary tournament selection with replacement, recombination and mutation for creating the offspring population.

Pareto Envelope-based Selection Algorithm II

Pareto Envelope-based Selection Algorithm II (PESA-II) [14, 28] employs a region-based selection, which enables to reduce the computational time for creating Pareto fronts. The search space is divided into hyper-boxes. Fitness functions assign every individual to a hyper-box. Using a standard selection method, a hyper-box is selected. Parents are randomly chosen from a given hyper-box and using standard genetic operators (crossover and mutation) offspring individuals are created. The selection algorithm operates with hyper-boxes instead of individuals.

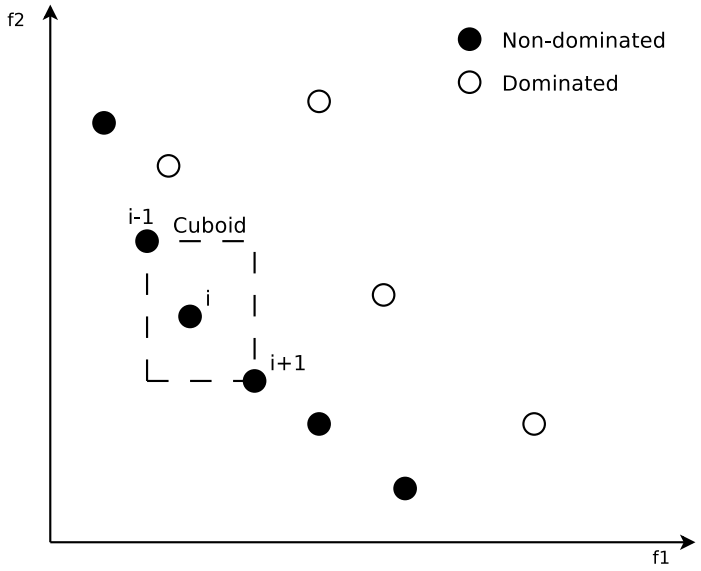

Figure 2.13: A cuboid used to determine the crowding distance of individual i.

Non-dominated Sorting Genetic Algorithm

One of the most popular multi-objective algorithms is Non-dominated Sorting Genetic Algorithm II (NSGA-II) proposed by Deb et al. in 2002 [17, 20, 37]. The algorithm is based on partitioning individuals from population P to non-dominated fronts. First front F_1 contains all non-dominated solutions. Every next front F_i is constructed as Pareto front for the population but individuals already included in F_{i-1}, F_{i-2}... are not considered. Each solution is assigned with a rank, which corresponds to the front ($p_{rank} = i$ for F_i). A naïve approach to create the non-dominated fronts requires $O(MN^3)$ operations, where M is the number of objectives and N is the population size.

The NSGA-II proposes a *fast-non-dominated sort*, which requires $O(MN^2)$ operations. In this algorithm, the set S_p contains individuals from the population that are dominated by individual p. The number of individuals which dominate p is stored in n_p. Each individual p in the first front has $n_p = 0$. Creating next fronts is based on knowledge of S_p and n_p. For each solution in F_i, we visit each member q from S_p and decrement the domination value

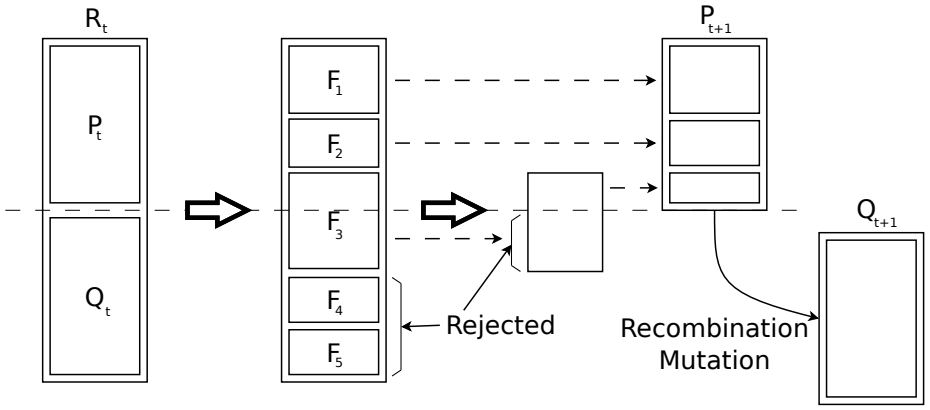

Figure 2.14: NSGA-II main algorithm step scheme.

$n_q = n_q - 1$. If any member gets $n_q = 0$, we put it to the next front F_{i+1}. This process continues until all fronts are identified.

The crowding distance assignment algorithm , differentiates individuals inside a front. The algorithm estimates the perimeter of the cuboid formed by the nearest neighbors to determine the crowding distance $i_{distance}$, see Figure 2.13. The value $i_{distance}$ is the average side length of the cuboid. The algorithm requires to sort individuals for each objective value. The boundary solutions are assigned with an infinite distance value. Other solutions are represented by a normalized distance of two nearby individuals. The crowding distance is the sum of these values for each objective. The normalization is computed using f_m^{min} and f_m^{max} values, which are the minimum and maximum values of m-th objective function.

The main steps of NSGA-II algorithm are shown on Figure 2.14. Parent population P_t and offspring population Q_t, both of size N, are combined to auxiliary population R_t and sorted using the fast non-dominated sorting algorithm. Each solution is assigned to a front. New parent population P_{t+1} is composed by adding individuals from fronts $F_1, F_2...$ until the number of individuals in the population is N. When the number of individuals in P_{t+1} exceeds N, the crowding distance algorithm is used to select additional individuals according to the distance of the parent population P_{t+1}. An offspring population Q_{t+1} is created from P_{t+1} using standard genetic operation: selection, recombination and mutation.

Non-dominated Sorting Genetic Algorithm III (NSGA-III) is a new version of NSGA-II intended for many-objective problems [19, 35]. A many-objective problem has more objectives than a multi-objective problem, typically more than five. The main difference is in the selection algorithm, where it is important to maintain the diversity of the population. NSGA-III employs a different strategy for including candidate individuals to the new population.

Chapter 3

Research Summary

This chapter summarizes the research process conducted in order to write this thesis. Chapter 3.1 introduces the methodology adopted to fulfill the objectives specified in Chapter 1. Chapter 3.2 presents selected papers of the author, their abstracts and contributions. Chapter 3.3 lists other papers of the author that are not included in this thesis.

3.1 Methodology

The overall objective addressed in this thesis is to improve key parameters of selected components of high-speed network monitoring systems. Based on our survey of the state-of-the-art approaches to network monitoring reported in Chapter 2, SDM and its hardware/software implementation developed in [45] has been chosen as a framework suitable for arranging and performing our experiments.

3.1.1 The Use of Evolutionary Computation Methods

The proposed approach is based on employing well-known GP algorithms; LGP for program design and CGP for circuit design and optimization. As it is necessary to optimize not only one parameter of the components (for example, the quality of processing), the proposed design/optimization approach has to consider more design objectives. In the thesis, two approaches have been developed:

- a single-objective approach based on constraining one of the objectives (e.g. by specifying the maximum acceptable latency) and optimizing the other objective (e.g. the quality) and
- a truly multi-objective method optimizing all design objectives together.

In order to accelerate the design process, a parallel LGP implementation has been developed and evaluated. Moreover, as computing the exact values of some component's parameters (e.g. the exact program execution time) is relatively time consuming, we had to cheaply estimate these values to obtain the appropriate fitness value. Because evolutionary algorithms are non-deterministic heuristics, their outcomes have to be statistically analyzed and interpreted. It has systematically been done in all the case studies reported in the papers constituting this thesis.

3.1.2 Selection of Target Components

In order to evaluate the proposed EA-based design and optimization approach, we selected two components – a circuit implementing a simplified application protocol classifier and a software hash function. We believe that these two components provide many properties and features that characterize the type of design problems that have to be addressed in systems such as SDM. The most interesting features are:

- We do not usually know a "perfect implementation" of these components in terms of functionality. The design of these components is usually based on experimental work whose objective is to minimize an error metric on various data sets.
- In both cases, performance (in other words, delay) has to be optimized in addition to the functionality.
- One of the components (the classifier) is implemented as a digital circuit; the second component (the hash function) is primarily implemented as a software routine. There is thus an opportunity to investigate if, in principle, the same methodology can be applied for their design and what are the differences.
- Hash functions are developed as either application-specific or general-purpose functions. There is an opportunity to investigate if one evolutionary design method can lead to acceptable results for both the scenarios.

3.1.3 Validation of Evolved Implementations

EAs require some training data sets in order to establish a fitness value. Other data sets are needed to validate the evolved solutions. In both our case studies, we used real network data collected by co-authors of our papers to evaluate and validate the evolved solutions. We also used additional real world data and synthetic data to evolve general-purpose hash functions. We implemented state-of-the-art classifiers and hash functions to compare the results they produce with evolved solutions. In the case of circuit implementations we employed industrial design tools for FPGAs to obtain the area and delay of evolved circuits. In the case of hash functions, the execution time was measured on common processors.

3.2 Papers

This chapter presents the papers included in this thesis. For each paper, we present an abstract, a brief description with motivation and a summary of the main contributions.

3.2.1 Paper I

GROCHOL David, SEKANINA Lukas, KORENEK Jan, ZADNIK Martin and KOSAR Vlastimil. Evolutionary Circuit Design for Fast FPGA-Based Classification of Network Application Protocols. Applied Soft Computing. Amsterdam: Elsevier Science, 2016, vol. 38, no. 1, pp. 933-941. ISSN 1568-4946.

Author participation: 40%
Journal Impact Factor (IF): 3.541

Abstract

The evolutionary design can produce fast and efficient implementations of digital circuits. It is shown in this paper how evolved circuits, optimized for the latency and area, can increase the throughput of a manually designed classifier of application protocols. The classifier is intended for high-speed networks operating at 100 Gbps. Because a very low latency is the main design constraint, the classifier is constructed as a combinational circuit in a field programmable gate array (FPGA). The classification is performed using the first packet carrying the application payload. The improvements in latency (and area) obtained by Cartesian genetic programming are validated using a professional FPGA design tool. The quality of classification is evaluated by means of real network data. All results are compared with commonly used classifiers based on regular expressions describing application protocols.

Contribution

In order to identify the application (or the application protocol) which the network traffic belongs to, one has to inspect one or several packets with a payload. The main difficulty is that the time to process one packet is less than 7 ns in the case of modern 100 Gbps links. This work is the extension of our initial work on the classifier design [30]. The main goal of the work is to show that these circuit classifiers can be optimized by means of Cartesian GP in order to reduce their latency and resources requirements. The improvements in latency and area obtained by CGP are validated by professional FPGA design tools. All results are compared with commonly used classifiers on several data sets.

This work introduces a new concept of the hardware classifier which is constructed as a fast-combinational circuit performing pattern matching over application protocols to be classified. We proposed accurate and relaxed versions of the classifier. Their circuit optimization by means of Cartesian GP led to 48.2% improvement in the area in FPGA (LUTs) and 19.8% improvement in latency with respect to an accurate human-designed classifier. Table 6 in Paper I shows results of synthesis for proposed classifiers. In order to compare the proposed solutions with the state-of-the-art classifiers from the literature, parameters of Yamagaki/Clark and AMTH circuit classifiers were included to this table. The classifiers were evaluated on real-network data.

3.2.2 Paper II

GROCHOL David and SEKANINA Lukas. Evolutionary Design of Fast High-quality Hash Functions for Network Applications. In: GECCO '16 Proceedings of the 2016 on Genetic and Evolutionary Computation Conference. New York, NY: Association for Computing Machinery, 2016, pp. 901-908. ISBN 978-1-4503-4206-3.

Author participation: 60%
Conference rank: A (Core)

Abstract

High-speed networks operating at 100 Gbps pose many challenges for hardware and software involved in the packet processing. As the time to process one packet is very short the corresponding operations have to be optimized in terms of the execution time. One of them is non-cryptographic hashing implemented in order to accelerate traffic flow identification.

In this paper, a method based on linear genetic programming is presented, which is capable of evolving high-quality hash functions primarily optimized for speed. Evolved hash functions are compared with conventional hash functions in terms of accuracy and execution time using real network data.

Contribution

One of the most frequently called functions in the flow processing is the hash function, which determines a memory address where the data of packet (flow) are stored. The goal of this work is to propose and evaluate a special hash function for flow hashing which has a good quality and is faster than the state-of-the-art hash functions. The hash function is constructed as a sequence of instructions for a CPU by means of a parallel linear GP exploiting the island model. In order to minimize the execution time, the hash function is constructed using a limited number of simple instructions. The evolved hash functions were compared with the hash functions available in the literature on real network datasets.

The paper shows that parallel single-objective LGP is capable of producing special hash functions for flow hashing. The program size is restricted to 12 instructions which was determined experimentally. Only simple instructions are used to minimize the execution time. The fitness function is based on the number of collisions and penalizing a solution generating many collisions on a given training data set. The evolved hash functions were compared with 11 hash functions available in the literature on real network data sets. The quality of hash functions is compared in Tab. 2 in Paper II. The best-evolved hash function has almost identical quality of hashing as the other hash functions but provides 3% improvement to the special network hash function (XORhash). Table 3 in Paper II compares the execution time of the hash functions on the CPU. The best-evolved hash function provides 26.9% improvement with the respect to the Murmur hash 3, which is typically used in SDM and which, on the other hand, provides a slightly lower number of collisions.

3.2.3 Paper III

GROCHOL David and SEKANINA Lukas. Multiobjective Evolution of Hash Functions for High Speed Networks. In: Proceedings of the 2017 IEEE Congress on Evolutionary Computation. San Sebastian: IEEE Computer Society, 2017, pp. 1533-1540. ISBN 978-1-5090-4600-3.

Author participation: 70%
Conference rank: B (Core)

Abstract

Hashing is a critical function in capturing and in an analysis of the network flows as its quality and execution time influences the maximum throughput of network monitoring devices. In this paper, we propose a multi-objective linear genetic programming approach to evolve fast and high-quality hash functions for common processors. The search algorithm simultaneously optimizes the quality of hashing and the execution time. As it is very time consuming to obtain the real execution time for a candidate solution on a particular processor, the execution time is estimated in the fitness function. In order to demonstrate the superiority of the proposed approach, evolved hash functions are compared with hash functions available in the literature using real-world network data.

Contribution

This work extends Paper II by including a multi-objective approach to the evolution process. The approach is based on the NSGA-II algorithm and linear GP. The multi-objective algorithm uses two fitness functions. The quality fitness function is taken from the previous work. The second fitness function estimates the execution time. Another contribution of this work is a new approach developed to quickly estimate the execution time of a candidate program. The execution time is estimated as a weighted number of instructions, where different weights are assigned to different types of instructions, based on their complexity. The estimation algorithm takes into account some features of modern CPUs, such as SIMD (Single Instruction Multiple Data) executions. The evolved hash functions were compared with hash functions available in the literature and hash functions obtained from our previous work.

This work resulted in an extension of LGP algorithm with a multi-objective approach. The quality of the execution time estimation is evaluated using randomly generated programs. The multi-objective method provided many non-dominated hash functions. Some of them are better than the commonly used hash functions and the specialized hash functions obtained by using the single-objective LGP with respect to chosen objective.

3.2.4 Paper IV

GROCHOL David and SEKANINA Lukas. Multi-Objective Evolution of Ultra-Fast General-Purpose Hash Functions. In: European Conference on Genetic Programming 2018. Berlin: Springer International Publishing, LNCS 10781, 2018, pp. 187-202. ISBN 978-3-319-77553-1.

Author participation: 70%
Conference rank: B (Core)

Abstract

Hashing is an important function in many applications such as hash tables, caches and Bloom filters. In the past, genetic programming was applied to evolve application-specific as well as general-purpose hash functions, where the main design target was the quality of hashing. As hash functions are frequently called in various time-critical applications, it is important to optimize their implementation with respect to the execution time. In this paper, linear genetic programming is combined with NSGA-II algorithm in order to obtain general-purpose, ultra-fast and high-quality hash functions. Evolved hash functions show a highly competitive quality of hashing but significantly reduced execution time in comparison with the state-of-the-art hash functions available in the literature.

Contribution

Paper II and Paper III have dealt with application-specific hash functions. This paper is focused on general-purpose hash functions that accept variable-length inputs, instead of a fixed-length input, which we considered in network hash functions. This change in the specification of hash functions led to the modification of the execution time estimation algorithm. Hence, the candidate (hash) program has to be wrapped to a loop, in which the input stream is processed block by block.

The evolved hash functions were compared with hash functions available in the literature on randomly generated data sets and real-world data sets (user passwords, network data,

Twitter and Facebook posts). The evolved hash functions produce a very similar number of collisions as other good hash functions from the literature on all data sets. However, evolved hash functions exhibit the shortest execution time in almost all cases on randomly generated and real-world data sets. They are slower than the special network hash functions, but faster than the general purpose hash functions when evaluated on the specific network datasets.

3.2.5 Paper V

GROCHOL David and SEKANINA Lukas. Fast Reconfigurable Hash Functions for Network Flow Hashing in FPGAs. In: Proceedings of the 2018 NASA/ESA Conference on Adaptive Hardware and Systems. Edinburgh: Institute of Electrical and Electronics Engineers, 2018, pp. 257-263. ISBN 978-1-5386-7753-7.

Author participation: 67%
Conference rank: unknown

Abstract

Efficient monitoring of high-speed computer networks operating with a 100 Gbps data throughput requires a suitable hardware acceleration of its key components. We present a platform capable of automated design of hash functions suitable for network flow hashing. The platform employs a multi-objective linear genetic programming developed for the hash function design. We evolved high-quality hash functions and implemented them in a FPGA. Several evolved hash functions were combined together in order to form the new reconfigurable hash function. The proposed reconfigurable design significantly reduces the area on a chip while the maximum operation frequency remains very close to the fastest hash functions. The characteristics of evolved hash functions were compared with the state-of-the-art hash functions in terms of the quality of hashing, chip area and the operation frequency in the FPGA.

Contribution

Using the methodology developed in Paper II and Paper III, we evolved hash functions suitable for FPGA implementations. We also introduced reconfigurable hash functions.

The evolved hash functions were translated to VHDL. In order to maximize their throughput, we added synchronization registers to enable pipelined processing. One of the reconfigurable hash functions was constructed using three evolved hash functions. These hash functions employ similar basic components that can be shared in the FPGA. The proposed reconfigurable hash function thus needs less than 50 % resources in comparison with the sum of resources needed to independently implement the three original hash functions.

3.3 List of Other Papers

- GROCHOL David, SEKANINA Lukas, ŽÁDNÍK Martin and KOŘENEK Jan. A Fast FPGA-Based Classification of Application Protocols Optimized Using Cartesian GP. In: Applications of Evolutionary Computation. Berlin: Springer International Publishing, LNCS 9028 , 2015, pp. 67-78. ISBN 978-3-319-16548-6.

  Author participation: 50%

Conference rank: unknown

- GROCHOL David. Evoluční hardware v síťových aplikacích. In: Počítačové architektury a diagnostika PAD 2016. Bořetice: Faculty of Information Technology BUT, 2016, pp. 57-60. ISBN 978-80-214-5376-0.

  Author participation: 100%

  Conference rank: unknown

- GROCHOL David and SEKANINA Lukas. Comparison of Parallel Linear Genetic Programming Implementations. In: Recent Advances in Soft Computing: Proceedings of the 22nd International Conference on Soft Computing (MENDEL 2016) held in Brno, Czech Republic, at June 8-10, 2016. Cham: Springer International Publishing, 2017, pp. 64-76. ISBN 978-3-319-58088-3.

  Author participation: 60%

  Conference rank: unknown

Chapter 4

Discussion and Conclusions

This chapter summarizes the results presented in this thesis and outlines some possibilities for a future research.

The research presented in this thesis was directed toward the optimization of selected components of network applications intended for high-speed network monitoring systems. The work started with a study of current network monitoring systems. As an experimental platform, the SDM system was chosen. Because the traffic processing is an important part of all monitoring systems, it was analyzed in a greater detail. For detailed studies conducted in this thesis two applications were selected: the classifier of application protocols and the hash functions for flow processing. The evolutionary computing techniques were surveyed with the aim to optimize not only the quality of processing, but also the execution time. The single-objective and multi-objective versions of evolution algorithms were considered. The gained knowledge was summarized in Chapter 2 and used as background for the following research.

The research started with the design and optimization of the application protocol classifier. As the SDM required an accurate classification of application protocols, the classifier was based on deep packet inspection (by means of the application data). The proposed application protocol classification is based on a pattern matching algorithm, which is a time-consuming operation, emphasizing the need for a hardware acceleration. The current approaches require a lot of resources in hardware. A new approach was proposed to classify a small set of protocols in the FPGA (denoted CL-acc in Paper I). The classifier was synthesized by a professional design tool to an FPGA. The final circuit of the classifier was optimized using CGP, reducing thus the amount of resources and latency. We also proposed relaxed implementations of the classifier. CL-cmp is a compromised version of the classifier (showing an additional area reduction for a small error in classification) and CL-lat is a minimal version of the classifier. Both relaxed classifiers were optimized by CGP. which enabled us to achieve a significant reduction of resources and latency. The accuracy of the classifiers was verified on real network data. On the other hand, FSM-based classifiers are more flexible and scalable.

The research continued with the hash function design using LGP. The first specialized network hash functions (evaluated for flow hashing) were optimized for the quality of hashing and constructed using a limited number of simple instructions. Single-objective and multi-objective LPG implementations were proposed for this purpose. Using the multi-objective approach, hash functions (NSGAHash1, NSGAHash2, NSGAHash3, NSGAHash4, NSGAHash5, NSGAHash6, NSGAHash7) were evolved, showing a better trade-off between the quality of hashing and the execution time than the state-of-the-art hash functions.

The pipelined versions of network hash functions were implemented for FPGA. An adaptive configurable hash function was also created from three evolved hash functions. Several high-quality general-purpose hash functions (EvoHash1, EvoHash2) were also evolved using the proposed method.

All evolved hash functions were evaluated on real-world network data sets (see NetSet1, NetSet2, NetSet3 in Paper V) and common real-world data sets (Passwords, Facebook, Twitter). The general-purpose hash functions were further evaluated on social network and randomly generated data sets. The evolved hash functions exhibit the same or better quality of hashing, but provide shorter execution time than the state-of-the-art hash functions.

4.1 Contributions

The section summarizes main contributions presented in this thesis, with respect to the research objectives formulated in Chapter 1.1:

Classification of application protocols:

- A new approach to the application protocol classifier design was proposed. Accurate and relaxed versions of the classifier were optimized by means of CGP. A significant reduction in FPGA resources and latency was reported in Paper I. A possible disadvantages of the proposed approach is that common classifiers are more flexible and scalable.

Hash Functions:

- Specialized, highly optimized network hash functions were evolved by parallel LGP. These hash functions provide better functionality (in terms of quality of hashing and execution time) than the state-of-the-art hash functions (Paper II).
- Using the multi-objective LGP, we evolved a set of non-dominated hash functions showing better trade-offs between the quality of network flow hashing and the execution time in comparison with the state-of-the-art hash functions (Paper III).
- Parallel pipelined hash functions were implemented in an FPGA and evaluated for purposes network flow hashing. A new reconfigurable hash function was developed as a combination of selected evolved hash functions (Paper V).
- Very competitive general-purpose hash functions were evolved by means of the multi-objective LGP and evaluated using representative data sets (Paper IV).

We also confirmed that common LGP a CGP implementations can be used for automated design and optimization of selected components; however, it is important to:

- properly handle the multi-objective nature of the problem and
- accelerate time-critical operations (particularly the fitness calculation).

Based on these results, it can be concluded that the initial hypothesis of this research has been confirmed. The proposed EAs can design and optimize selected components of network applications of high-speed network monitoring systems and improve their key parameters.

4.2 Future Work

Based on our experience gained during this research the following future research directions were identified:

- Network monitoring systems are large and complex systems composed of many components. It is not a straightforward task to identify the critical components that should be optimized. Automated identification of such components for evolutionary re-design/optimization would be of high importance.
- An automated runtime optimization of components would be useful because the monitoring system could be adapted to the actual state of the system. If the optimization is fast, the component can be optimized for a specific situation or an important subset of input data.
- If the automated identification of components in network monitoring systems is connected to the runtime optimization, the system could adapt different components in runtime in a variable environment.
- Modern CPUs utilize many complex instructions, including application-specific instructions, such as hash function (for example Intel CPU: SHA1RNDS4 or AESDECLAST), special floating-point instructions or SIMD instructions (MMX and SSE). A future research could be focused on identifying a suitable subset of instructions that can be utilized by LGP; considering all possible instructions in LGP seems to be intractable.
- Other HW parts of monitoring systems those implemented in FPGA can be optimized using a multi-objective CGP. Contrasted to our work based on gate-level circuit optimization, a future work could deal with LUT-based circuit optimization in order to obtain more efficient FPGA implementations.

Bibliography

[1] Antichi, G.; Giordano, S.; Miller, D.; et al.: Enabling open-source high speed network monitoring on NetFPGA. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*. April 2012. pp. 1029–1035.

[2] Appleby, A.: Murmur hash functions. Https://github.com/aappleby/smhasher, [ONLINE, accessed: 31. 1. 2016].

[3] Appleby, A.: SMHasher. Https://github.com/aappleby/smhasher, [ONLINE, accessed: 1. 11. 2017].

[4] Becker, K.; Gottschlich, J.: AI Programmer: Autonomously Creating Software Programs Using Genetic Algorithms. *arXiv preprint arXiv:1709.05703*. 2017.

[5] Berarducci, P.; Jordan, D.; Martin, D.; et al.: GEVOSH: Using Grammatical Evolution to Generate Hashing Functions. In *MAICS*. 2004. pp. 31–39.

[6] Bernstein, D. J.: Mathematics and computer science. Https://cr.yp.to/djb.html, [ONLINE, accessed: 31. 1. 2016].

[7] Brameier, M.; Banzhaf, W.: *Linear genetic programming*. New York: Springer. 2007.

[8] Brodie, B. C.; Taylor, D. E.; Cytron, R. K.: A Scalable Architecture For High-Throughput Regular Expression Pattern Matching. *SIGARCH Computer Architecture News*. vol. 34, no. 2. 2006: pp. 191–202. ISSN 0163-5964.

[9] Cao, Z.; Wang, Z.: Flow identification for supporting per-flow queueing. In *Computer Communications and Networks, 2000. Proceedings. Ninth International Conference on*. IEEE. 2000. pp. 88–93.

[10] Cheang, S. M.; Lee, K. H.; Leung, K. S.: Applying Genetic Parallel Programming to Synthesize Combinational Logic Circuits. *IEEE Transactions on Evolutionary Computation*. vol. 11, no. 4. 2007: pp. 503–520.

[11] Cisco: The Future Is 40 Gigabit Ethernet. 2016. c11-737238-00.

[12] Clark, C.; Schimmel, D.: Efficient Reconfigurable Logic Circuits for Matching Complex Network Intrusion Detection Patterns. In *Field Programmable Logic and Application, 13th International Conference*. Lisbon, Portugal. 2003. ISBN 3-540-40822-3. pp. 956–959.

[13] Clark, C. R.; Schimmel, D. E.: Scalable Pattern Matching for High-Speed Networks. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. Napa, California. 2004. pp. 249–257.

[14] Corne, D. W.; Jerram, N. R.; Knowles, J. D.; et al.: PESA-II: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc.. 2001. pp. 283–290.

[15] Davis, L.: Handbook of genetic algorithms. 1991.

[16] Deb, K.: *Multi-objective optimization using evolutionary algorithms*. vol. 16. John Wiley & Sons. 2001.

[17] Deb, K.; Agrawal, S.; Pratap, A.; et al.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *International Conference on Parallel Problem Solving From Nature*. Springer. 2000. pp. 849–858.

[18] Deb, K.; Deb, K.: *Multi-objective Optimization*. Boston, MA: Springer US. 2014. ISBN 978-1-4614-6940-7. pp. 403–449.

[19] Deb, K.; Jain, H.: An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints. *IEEE Transactions on Evolutionary Computation*. vol. 18, no. 4. Aug 2014: pp. 577–601. ISSN 1089-778X.

[20] Deb, K.; Pratap, A.; Agarwal, S.; et al.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*. vol. 6, no. 2. 2002: pp. 182–197.

[21] Defoin Platel, M.; Clergue, M.; Collard, P.: Maximum Homologous Crossover for Linear Genetic Programming. In *Genetic Programming*, *Lecture Notes in Computer Science*, vol. 2610. Springer Berlin Heidelberg. 2003. ISBN 978-3-540-00971-9. pp. 194–203.

[22] Downey, C.; Zhang, M.; Browne, W. N.: New crossover operators in linear genetic programming for multiclass object classification. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM. 2010. pp. 885–892.

[23] Ehrgott, M.: *Multicriteria optimization*. vol. 491. Springer Science & Business Media. 2005.

[24] Estebanez, C.; Saez, Y.; Recio, G.; et al.: Automatic design of noncryptographic hash functions using genetic programming. *Computational Intelligence*. vol. 30, no. 4. 2014: pp. 798–831.

[25] Filtr, L.: Project WWW Page. `http://l7-filter.sourceforge.net/`. 2010.

[26] Floyd, R. W.; Ullman, J. D.: The Compilation of Regular Expressions into Integrated Circuits. *J. ACM*. vol. 29, no. 3. 1982: pp. 603–622.

[27] Fowler, G.; Vo, P.; Noll, L. C.: FVN Hash. Http://www.isthe.com/chongo/tech/comp/fnv/, [ONLINE, accessed: 31. 1. 2016].

[28] Gadhvi, B.; Savsani, V.; Patel, V.: Multi-objective optimization of vehicle passive suspension system using NSGA-II, SPEA2 and PESA-II. *Procedia Technology*. vol. 23. 2016: pp. 361–368.

[29] Goldman, B. W.; Punch, W. F.: Analysis of Cartesian Genetic Programming's Evolutionary Mechanisms. *IEEE Transactions on Evolutionary Computation*. vol. 19, no. 3. 2015: pp. 359–373.

[30] Grochol, D.; Sekanina, L.; Zadnik, M.; et al.: A Fast FPGA-Based Classification of Application Protocols Optimized Using Cartesian GP. In *Applications of Evolutionary Computation, 18th European Conference*. LNCS 9028. Springer International Publishing. 2015. pp. 67–78.

[31] Guo, D.; Bhuyan, L. N.; Liu, B.: An efficient parallelized L7-filter design for multicore servers. *IEEE/ACM Transactions on Networking*. vol. 20, no. 5. 2011: pp. 1426–1439.

[32] Guo, D.; Liao, G.; Bhuyan, L. N.; et al.: A scalable multithreaded l7-filter design for multi-core servers. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ACM. 2008. pp. 60–68.

[33] Hassan, M.; Jain, R.: *High performance TCP/IP networking*. vol. 29. Prentice Hall Upper Saddle River, NJ. 2003.

[34] Higuchi, T.; Niwa, T.; Tanaka, T.; et al.: Evolving Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine. In *Proc. of the 2nd International Conference on Simulated Adaptive Behaviour*. MIT Press. 1993. pp. 417–424.

[35] Jain, H.; Deb, K.: An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach. *IEEE Transactions on Evolutionary Computation*. vol. 18, no. 4. Aug 2014: pp. 602–622. ISSN 1089-778X.

[36] Jenkins, B.: A hash function for hash Table lookup. Http://www.burtleburtle.net/bob/hash/doobs.html, [ONLINE, accessed: 31. 1. 2016].

[37] Jensen, M. T.: Reducing the run-time complexity of multiobjective EAs: The NSGA-II and other algorithms. *IEEE Transactions on Evolutionary Computation*. vol. 7, no. 5. 2003: pp. 503–515.

[38] Karagiannis, T.; Papagiannaki, K.; Faloutsos, M.: BLINC: Multilevel Traffic Classification in the Dark. *SIGCOMM Comput. Commun. Rev.*. vol. 35, no. 4. 2005: pp. 229–240.

[39] Karasek, J.; Burget, R.; Morskỳ, O.: Towards an automatic design of non-cryptographic hash function. In *Telecommunications and Signal Processing (TSP), 2011 34th International Conference on*. 2011. pp. 19–23.

[40] Kaufmann, P.; Glette, K.; Gruber, T.; et al.: Classification of Electromyographic Signals: Comparing Evolvable Hardware to Conventional Classifiers. *IEEE Tran. Evolutionary Computation,*. vol. 17, no. 1. 2013: pp. 46–63.

[41] Kaufmann, P.; Plessl, C.; Platzner, M.: EvoCaches: Application-specific Adaptation of Cache Mappings. In *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE Computer Society. 2009. pp. 11–18.

[42] Kaufmann, P.; Plessl, C.; Platzner, M.: EvoCaches: Application-specific Adaptation of Cache Mappings. In *Adaptive Hardware and Systems (AHS)*. IEEE CS. 2009. pp. 11–18.

[43] Kekely, L.; Kucera, J.; Pus, V.; et al.: Software Defined Monitoring of Application Protocols. *IEEE Transactions on Computers*. vol. 65, no. 2. 2016: pp. 615–626.

[44] Kekely, L.; Pus, V.; Benacek, P.; et al.: Trade-offs and progressive adoption of FPGA acceleration in network traffic monitoring. In *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*. 2014. pp. 1–4.

[45] Kekely, L.; Pus, V.; Korenek, J.: Software Defined Monitoring of Application Protocols. In *Proceedings of the IEEE INFOCOM 2014 — IEEE Conference on Computer Communications*. 2014. pp. 1725–1733.

[46] Kekely, M.; Kořenek, J.: Packet Classification with Limited Memory Resources. In *In proceedings 2017 Euromicro Conference on Digital System Design*. Institute of Electrical and Electronics Engineers. 2017. ISBN 978-1-5386-2145-5. pp. 179–183.

[47] Kidoň, M.; Dobai, R.: Evolutionary design of hash functions for IP address hashing using genetic programming. In *Evolutionary Computation (CEC), 2017 IEEE Congress on*. IEEE. 2017. pp. 1720–1727.

[48] King, R. A.; Deb, K.; Rughooputh, H.: Comparison of nsga-ii and spea2 on the multiobjective environmental/economic dispatch problem. *University of Mauritius Research Journal*. vol. 16, no. 1. 2010: pp. 485–511.

[49] Knowles, J.; Corne, D.: On metrics for comparing nondominated sets. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, vol. 1. IEEE. 2002. pp. 711–716.

[50] Knuth, D. E.: *The Art of Computer Programming (Volume 3)*. 1973.

[51] Kocsis, Z. A.; Neumann, G.; Swan, J.; et al.: Repairing and optimizing Hadoop hashCode implementations. In *International Symposium on Search Based Software Engineering*. Springer. 2014. pp. 259–264.

[52] Koza, J. R.: Genetic programming as a means for programming computers by natural selection. *Statistics and computing*. vol. 4, no. 2. 1994: pp. 87–112.

[53] Koza, J. R.: Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*. vol. 11, no. 3-4. 2010: pp. 251–284.

[54] Landfeldt, B.; Sookavatana, P.; Seneviratne, A.: The case for a hybrid passive/active network monitoring scheme in the wireless Internet. In *Proceedings IEEE International Conference on Networks 2000 (ICON 2000). Networking Trends and Challenges in the New Millennium*. Sep. 2000. pp. 139–143. doi:10.1109/ICON.2000.875781.

[55] Langdon, W. B.; Harman, M.: Optimizing existing software with genetic programming. *IEEE Transactions on Evolutionary Computation*. vol. 19, no. 1. 2015: pp. 118–135.

[56] Langdon, W. B.; Lam, B. Y. H.; Modat, M.; et al.: Genetic improvement of GPU software. *Genetic Programming and Evolvable Machines*. vol. 18, no. 1. 2017: pp. 5–44.

[57] Liao, G.; Znu, X.; Bnuyan, L.: A new server I/O architecture for high speed networks. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. 2011. ISSN 2378-203X. pp. 255–265. doi:10.1109/HPCA.2011.5749734.

[58] Lin, C.-H.; Huang, C.-T.; Jiang, C.-P.; et al.: Optimization of Pattern Matching Circuits for Regular Expression on FPGA. *IEEE Trans. Very Large Scale Integr. Syst.*. vol. 15, no. 12. 2007: pp. 1303–1310. ISSN 1063-8210.

[59] Matousek, D.; Matousek, J.; Korenek, J.: High-Speed Regular Expression Matching with Pipelined Memory-Based Automata. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. April 2018. ISSN 2576-2621. pp. 214–214.

[60] Matoušek, D.; Kubiš, J.; Matoušek, J.; et al.: Regular Expression Matching with Pipelined Delayed Input DFAs for High-speed Networks. 07 2018. pp. 104–110. doi:10.1145/3230718.3230730.

[61] Maurer, W. D.; Lewis, T. G.: Hash table methods. *ACM Computing Surveys (CSUR)*. vol. 7, no. 1. 1975: pp. 5–19.

[62] Miller, J. F.: *Cartesian Genetic Programming*. Springer-Verlag. 2011.

[63] Miller, J. F.; Smith, S. L.: Redundancy and Computational Efficiency in Cartesian Genetic Programming. *IEEE Transactions on Evolutionary Computation*. vol. 10, no. 2. 2006: pp. 167–174.

[64] Miller, J. F.; Thomson, P.: Cartesian Genetic Programming. In *Proc. of the 3rd European Conference on Genetic Programming EuroGP2000*, *LNCS*, vol. 1802. Springer. 2000. pp. 121–132.

[65] Moore, A. W.; Zuev, D.: Internet Traffic Classification Using Bayesian Analysis Techniques. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS '05. ACM. 2005. pp. 50–60.

[66] Natu, M.; Sethi, A. S.: Active Probing Approach for Fault Localization in Computer Networks. In *2006 4th IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services*. April 2006. pp. 25–33. doi:10.1109/E2EMON.2006.1651276.

[67] Oltean, M.; Grosan, C.: A comparison of several linear genetic programming techniques. *Complex Systems*. vol. 14, no. 4. 2003: pp. 285–314.

[68] Pagh, R.; Rodler, F. F.: Cuckoo Hashing. In *Algorithms — ESA 2001*. LNCS 2161. Springer. 2001. pp. 121–133.

[69] Paxson, V.; Asanović, K.; Dharmapurikar, S.; et al.: Rethinking Hardware Support for Network Analysis and Intrusion Prevention. In *Proceedings of the 1st USENIX Workshop on Hot Topics in Security*. HOTSEC'06. Berkeley, CA, USA: USENIX Association. 2006. pp. 11–11.
Retrieved from: http://dl.acm.org/citation.cfm?id=1268476.1268487

[70] Peterson, L. L.; Davie, B. S.: *Computer networks: a systems approach*. Elsevier. 2007.

[71] Pike, G.; Alakuijala, J.: Introducing cityhash. 2011.

[72] Press, C.: *CCNA Exploration Course Booklet: Network Fundamentals, Version 4.0*. Pearson Education India.

[73] Price, K.; Storn, R.: Differential evolution: A simple evolution strategy for fast optimization. *Dr. Dobb's journal*. vol. 22, no. 4. 1997: pp. 18–24.

[74] Rechenberg, I.: Evolution Strategy: Optimization of Technical systems by means of biological evolution. *Fromman-Holzboog, Stuttgart*. vol. 104. 1973: pp. 15–16.

[75] Rozenberg, G.; Bäck, T.; Kok, J. N.: *Handbook of natural computing*. Springer. 2012.

[76] Safdari, M.; Joshi, R.: Evolving Universal Hash Functions Using Genetic Algorithms. In *In Proc. of the Future Computer and Communication*. 2009. pp. 84–87.

[77] Schaffer, J.: Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. 01 1985. pp. 93–100.

[78] Schwaller, P. J.; Bellinghausen, J. M.; Borger, D. S.; et al.: Methods, systems and computer program products for network performance testing through active endpoint pair based testing and passive application monitoring. September 23 2003. uS Patent 6,625,648.

[79] Sekanina, L.: Evolvable hardware. In *Handbook of Natural Computing*. Springer Verlag. 2012. pp. 1657–1705.

[80] Sen, S.; Spatscheck, O.; Wang, D.: Accurate, Scalable In-network Identification of P2P Traffic Using Application Signatures. In *Proceedings of the 13th International Conference on World Wide Web*. ACM. 2004. pp. 512–521.

[81] Shanthi, A. P.; Parthasarathi, R.: Practical and scalable evolution of digital circuits. *Applied Soft Computing*. vol. 9, no. 2. 2009: pp. 618–624.

[82] Sidhu, R.; Prasanna, V. K.: Fast Regular Expression Matching Using FPGAs. In *FCCM '01: Proceedings of the the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. IEEE Computer Society. 2001. ISBN 0-7695-2667-5. pp. 227–238.

[83] Sourdis, I.; Bispo, J.; Cardoso, J. M. P.; et al.: Regular Expression Matching in Reconfigurable Hardware. *Journal of Signal Processing Systems*. vol. 51, no. 1. 2008: pp. 99–121.

[84] Srivani, L.; Giri, N. K.; Ganesh, S.; et al.: Generating synthetic benchmark circuits for accelerated life testing of field programmable gate arrays using genetic algorithm and particle swarm optimization. *Applied Soft Computing*. vol. 27. 2015: pp. 179 – 190.

[85] Stallings, W.: *High-speed networks: TCP/IP and ATM design principles*. vol. 172. Prentice hall Englewood Cliffs, NJ. 1998.

[86] Standard, S. H.: The Cryptographic Hash Algorithm Family: Revision of the Secure Hash Standard and Ongoing Competition for New Hash Algorithms. 2009.

[87] Stomeo, E.; Kalganova, T.; Lambert, C.: Generalized Disjunction Decomposition for Evolvable Hardware. *IEEE Transaction Systems, Man and Cybernetics, Part B.* vol. 36, no. 5. 2006: pp. 1024–1043.

[88] Vasicek, Z.; Bidlo, M.; Sekanina, L.: Evolution of efficient real-time non-linear image filters for FPGAs. *Soft Computing.* vol. 17, no. 11. 2013: pp. 2163–2180.

[89] Vasicek, Z.; Sekanina, L.: Formal Verification of Candidate Solutions for Post-Synthesis Evolutionary Optimization in Evolvable Hardware. *Genetic Programming and Evolvable Machines.* vol. 12, no. 3. 2011: pp. 305–327.

[90] Walker, J. A.; Trefzer, M.; Bale, S. J.; et al.: PAnDA: A Reconfigurable Architecture that Adapts to Physical Substrate Variations. *IEEE Transactiona on Computers.* vol. 62, no. 8. 2013: pp. 1584–1596.

[91] Widiger, H.; Salomon, R.; Timmermann, D.: Packet classification with evolvable hardware hash functions–an intrinsic approach. In *International Workshop on Biologically Inspired Approaches to Advanced Information Technology.* Springer. 2006. pp. 64–79.

[92] Wilson, G.; Banzhaf, W.: A comparison of cartesian genetic programming and linear genetic programming. In *Genetic Programming.* Springer. 2008. pp. 182–193.

[93] Xilinx: UltraScale Architecture and Product Overview. 2015.

[94] Xilinx Inc.: UltraScale+ FPGA, Product Tables and Product Selection Guide. Https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf, [ONLINE, accessed: 26. 7. 2019].

[95] Yoon, S.-H.; Park, J.-W.; Park, J.-S.; et al.: Internet Application Traffic Classification Using Fixed IP-Port. In *APNOMS*, *Lecture Notes in Computer Science*, vol. 5787. Springer. 2009. pp. 21–30.

[96] Yun, S.; Lee, K.: Optimization of Regular Expression Pattern Matching Circuit Using At-Most Two-Hot Encoding on FPGA. *International Conference on Field Programmable Logic and Applications.* vol. 0. 2010: pp. 40–43. ISSN 1946-1488.

[97] Zimmermann, H.: OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications.* vol. 28, no. 4. April 1980: pp. 425–432. ISSN 0090-6778. doi:10.1109/TCOM.1980.1094702.

[98] Zitzler, E.; Laumanns, M.; Thiele, L.: SPEA2: Improving the strength Pareto evolutionary algorithm. *TIK-report.* vol. 103. 2001.

Curriculum vitae

Ing. David Grochol

Education

2014–yet **Ph.D. of Computer Science and Engineering** , *Faculty of Information Technology, Brno University of Technology* , Brno.
Supervisor: prof. Ing. Lukáš Sekanina, Ph.D.

2012–2014 **Master of Bioinformatics and Biocomputing**, *Faculty of Information Technology, Brno University of Technology* , Brno, *Ing.*
Master thesis: Fast Detection of Application Protocols,
Supervisor: prof. Ing. Lukáš Sekanina, Ph.D.

2009–2012 **Bachelor of Information Technology**, *Faculty of Information Technology, Brno University of Technology* , Brno, *Bc.*
Bachelor thesis: Document Archiving System,
Supervisor: Ing. Jiří Ševcovic

2005–2009 **Secondary school**, *Secondary Technical School Trinec*, Trinec.
Automation technology

Conferences

2015 Evostar

2016 GECCO, MENDEL, PAD

2017 CEC

2018 EuroGP, AHS

Projects

- GA14-04197S - Advanced Methods for Evolutionary Design of Complex Digital Circuits
- FIT-S-14-2297 - Architecture of parallel and embedded computer systems
- GA16-08565S - Advancing cryptanalytic methods through evolutionary computing
- LQ1602 - IT4Innovations excellence in science
- FIT-S-17-3994 - Advanced parallel and embedded computer systems
- LTC18053 - Advanced Methods of Nature-Inspired Optimisation and HPC Implementation for the Real-Life Applications

Teaching

- Practical Aspects of Software Design – *labs*
- Personal Computers – *lectures, projects*
- Bio-Inspired Computers – *labs*

Publications

GROCHOL David, SEKANINA Lukas, ŽÁDNÍK Martin and KOŘENEK Jan. **A Fast FPGA-Based Classification of Application Protocols Optimized Using Cartesian GP**. In: *Applications of Evolutionary Computation*. Berlin: Springer International Publishing, LNCS 9028 , 2015, pp. 67-78. ISBN 978-3-319-16548-6.

Author participation: 50%
Conference rank: unknown
Cited: WoS: 0, Scopus: 0

GROCHOL David. **Evoluční hardware v síťových aplikacích**. In: *Počítačové architektury a diagnostika PAD 2016*. Bořetice: Faculty of Information Technology BUT, 2016, pp. 57-60. ISBN 978-80-214-5376-0.

Author participation: 100%
Conference rank: unknown
Cited: WoS: 0, Scopus: 0

GROCHOL David, SEKANINA Lukas, KORENEK Jan, ZADNIK Martin and KOSAR Vlastimil. **Evolutionary Circuit Design for Fast FPGA-Based Classification of Network Application Protocols**. *Applied Soft Computing*. Amsterdam: Elsevier Science, 2016, vol. 38, no. 1, pp. 933-941. ISSN 1568-4946.

Author participation: 40%
Journal Impact Factor (IF): 3.541
Cited: WoS: 0, Scopus: 0

GROCHOL David and SEKANINA Lukas. **Evolutionary Design of Fast High-quality Hash Functions for Network Applications**. In: *GECCO '16 Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. New York, NY: Association for Computing Machinery, 2016, pp. 901-908. ISBN 978-1-4503-4206-3.

Author participation: 60%
Conference rank: A (Core)
Cited: WoS: 3, Scopus: 3

GROCHOL David and SEKANINA Lukas. **Comparison of Parallel Linear Genetic Programming Implementations**. In: *Recent Advances in Soft Computing: Proceedings of the 22nd International Conference on Soft Computing (MENDEL 2016) held in Brno, Czech Republic, at June 8-10, 2016*. Cham: Springer International Publishing, 2017, pp. 64-76. ISBN 978-3-319-58088-3.

Author participation: 60%
Conference rank: unknown
Cited: WoS: 0, Scopus: 0

GROCHOL David and SEKANINA Lukas. **Multiobjective Evolution of Hash Functions for High Speed Networks**. In: *Proceedings of the 2017 IEEE Congress on Evolutionary Computation*. San Sebastian: IEEE Computer Society, 2017, pp. 1533-1540. ISBN 978-1-5090-4600-3.

Author participation: 70%
Conference rank: B (Core)
Cited: WoS: 0, Scopus: 0

GROCHOL David and SEKANINA Lukas. **Multi-Objective Evolution of Ultra-Fast General-Purpose Hash Functions**. In: *European Conference on Genetic Programming 2018*. Berlin: Springer International

Publishing, LNCS 10781, 2018, pp. 187-202. ISBN 978-3-319-77553-1.

Author participation: 70%
Conference rank: B (Core)
Cited: WoS: 0, Scopus: 0

GROCHOL David and SEKANINA Lukas. **Fast Reconfigurable Hash Functions for Network Flow Hashing in FPGAs**. In: *Proceedings of the 2018 NASA/ESA Conference on Adaptive Hardware and Systems*. Edinburgh: Institute of Electrical and Electronics Engineers, 2018, pp. 257-263. ISBN 978-1-5386-7753-7.

Author participation: 67%
Conference rank: unknown
Cited: WoS: 0, Scopus: 0

Work Experience

2017–yet **Developer**, *ARTISYS s.r.o,*, Brno.
Software Developer

2012–2017 **Operator**, *CBL Communication by light s.r.o.*, Brno.
Network Monitoring and Troubleshooting

2016–2017 **Operator**, *Miracle Network*, Brno.
Network Monitoring and Troubleshooting

2010–2012 **Operator**, *STAR 21 Networks, a.s.*, Brno.
Network Monitoring and Troubleshooting