A Fast FPGA-Based Classification of Application Protocols Optimized Using Cartesian GP

David Grochol, Lukas Sekanina, Martin Zadnik, and Jan Korenek

Brno University of Technology, Faculty of Information Technology,
IT4Innovations Centre of Excellence
Božetěchova 2, 612 66 Brno, Czech Republic
igrochol@fit.vutbr.cz, sekanina@fit.vutbr.cz, izadnik@fit.vutbr.cz,
korenek@fit.vutbr.cz

Abstract. This paper deals with design of an application protocol classifier 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. The classification is performed using the first packet carrying the application payload. In order to further reduce the latency, the circuit is optimized by Cartesian genetic programming. Using a real network data, we demonstrated viability of our approach in task of a very fast classification of three application protocols (HTTP, SMTP, SSH).

1 Introduction

An abstract yet detailed network traffic visibility is a key prerequisite to network management including tasks such as traffic engineering, application performance monitoring and network security monitoring. In the recent years the diversity and complexity of network applications and network threats have grown significantly. This trend has rendered monitoring of network and transport layer insufficient and it has become important to extend the visibility into application layer, primarily to identify the application (or the application protocol) the traffic belongs to. The port numbers are no longer reliable application differentiator due to various applications evading the firewalls by hiding behind well-known port numbers or utilizing unallocated port range [1].

The research in the area of application identification has come up with distinct approaches to identify applications carried in the traffic. These approaches differ in the level of detail that is utilized in the identification method. The most abstract one is behavioral analysis [2, 3]. Its idea is to observe just port number 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, statistical fingerprinting [4] 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 they leave a unique fingerprint. Behavioral and statistical fingerprinting generally classifies traffic to application classes rather than to the particular applications. The reason is that different applications performing the same task exhibit similar behavior. For instance, application protocols such as Oscar (ICQ), MSN, XMPP (Jabber) transport interactive chat communications and hence have a similar behavior, which makes it very hard to differentiate between them. The inability to distinguish applications within the same class might be seen as a drawback in some situations when, for example, it is necessary to block a particular application while allowing others in the same class. The approach utilizing the greatest level of detail is a deep packet inspection. It identifies applications based on the packet payload. The payload is matched with known patterns (e.g. regular expressions) derived for each application [5].

The application identification poses several on-going challenges. The identification process is bound to keep pace with ever increasing link speeds. E.g. the time to process each packet is less than 7 ns in case of 100 Gbps link. Another challenge is represented by the growing number of protocols, i.e., the application identification must address the trends such as new emerging mobile applications or applications moving into network cloud [6]. Some deployments also require prompt (near real-time) identification to enable implementation of traffic engineering or application blocking [7].

Hardware acceleration (e.g. utilizing a field programmable gate array (FPGA)) is often employed to speed up network processing [8, 9] including the application identification directly on the network card. FPGA renders it possible to utilize various pattern matching algorithms to identify applications. However, pattern matching may exhibit several constraints, that is, the high cost to process wide data inputs (which is the case for high throughput buses in FPGA) and the high complexity and overhead of pattern matching algorithm which consumes valuable hardware resources or constraints the achievable frequency.

These drawbacks are addressed by alternative methods which look for constants and fixed-length strings (for brevity we call them signatures) rather than regular expressions, e.g. [10]. We build upon this approach and we envision hardware-software codesign approach in which a simple circuit labels the traffic belonging to applications of interest with some probability of false positives while software can subsequently handle and check the labeled traffic with more complex algorithm effectively. This approach is supported by the software defined monitoring concept (SDM, [11]). Software defined monitoring employs sophisticated processes running in software to subsequently install rules in the hardware (network card). While it is not possible (or at the very high cost) to process all the traffic in software we offload the application identification into the hardware. The offload not only reduces the host memory and cpu load but it also increases the expressive strength of the SDM rules.

Within this scope, our work focuses on a design of a proprietary circuit, operating as an application protocol classifier, which is synthesizable into FPGA. The goal of our work is to identify application protocols as fast as possible over the whole traffic but at the same time with low resource utilization. We demonstrate viability of our approach on three protocols (HTTP, SMTP, SSH) we deem most crucial from perspective of network security. The proposed circuit, in fact, implements a deterministic parallel combinational signature matching algorithm.

The main contribution of this paper is showing that this circuit classifier can be optimized by means of Cartesian Genetic Programming (CGP) in order to significantly reduce its latency and resources requirements. CGP, which is a form of genetic programming suitable for evolutionary circuit design and optimization, is employed to minimize the number of gates in selected components of the whole classifier [12]. The improvements in latency and area obtained by CGP are validated using a professional FPGA design tool. The quality of classification is evaluated by means of real network data.

The rest of the paper is organized as follows. Section 2 introduces the proposed classifier, network data used for its evaluation and the principles of circuit optimization by means of CGP. Section 3 describes the implementation steps and the results in terms of area and delay in the FPGA. Section 3.2 deals with the experiments conducted using CGP. Finally, the quality of classification is assessed in terms of precision and recall. Conclusions are given in Section 4.

2 FPGA-based Application Protocol Classifier

In order to design, implement and evaluate the FPGA-based application protocol classifier, we have considered the following issues. As this is the first version of the classifier, only 3 application protocols (HTTP, SMTP, SSH) will be supported; remaining protocols will be classified as unknown. Only the key component of the classifier (9 coders with a comparator) will be synthesized and optimized for FPGA. This will provide us with a sufficiently precise estimate of the latency and area of the whole classifier. Because the primary goal is achieving a very low latency, only the signatures of the first packet carrying the application payload are utilized. The classifier is intended for SDM system which transfers the data via a 512 bit bus. The application payload may start at nearly arbitrary offset on the bus and the application must be identified each clock cycle to keep pace even with the shortest incoming packets of 64 Bytes. The following subsections describe the whole design process and methods utilized.

2.1 Network Data

The data which has to be classified are common network data (available in the pcap format). In order to design a good solution, it is important to understand the data and to prepare relevant training and test sets. In our case, we utilized two complete network data sets with anonymized IP addresses, collected on CESACO link (connecting CESNET and ACONET networks) and CESPIO link (connecting CESNET and PIONIER networks), see Table 1. For example, the available record from CESPIO contains 43 M packets, where percentages are 78.72% for TCP, 20.58% for UDP, 0.18% for ICMP and 0.53% others. One can observe that only TCP and UDP are relevant for our purposes; ICMP is used for network monitoring and error reporting which is irrelevant. The packet traces were analyzed using Scapy. Because HTTP, SMTP and SSH operate over TCP, we consider the first packet containing the application payload, usually the third or the fourth packet of the TCP connection. The L7 filter [13] was utilized as a reference classifier to anotate each connection in the data set. Unfortunately, implementing the L7 filter, based on regular expressions, in hardware would lead to a high latency as well as resource utilization which is not acceptable in our case. Other packets were labeled as unknown.

Table 1. Network data sets

Line	Speed [Gb/s]	Record duration	Size [Gb]	Date
CESACO	10	9 s	1	26.9.2013 10:52:02
CESPIO	10	8 min	35	26.2.2014 11:38:56

The resulting data set, which can be used for evaluation purposes, is available in the JSON format. Each record contains the source IP and port, the destination IP and port, the transport protocol number, and the whole packet encoded using base64 (see Fig. 1). Table 2 gives the mix of considered protocols in our data set.

Table 2. The flows corresponding to the application protocols in data sets CESACO and CESPIO

Data set	CESACO		CESPIO	
Protocol	Count flows	Count flows [%]	Count flows	Count flows [%]
HTTP	1914	38.12	15060	52.29
SMTP	4	0.08	34	0.12
SSH	1	0.02	0	0.00
Others	3102	61.78	13705	47.59
All	5021	100.00	28799	100.00

2.2 Deterministic Classifier

Because the classification utilizes only the start of the payload, we decided to analyze several initial bytes of various commonly used application protocols and find the characters which are unique for particular protocols. Table 3 shows the unique character *signatures* that were identified for considered protocols (the longest sequence is 9 bytes). The classifier can then be constructed as a combinational circuit by means of a decoder. However, it has to correctly manage the cases in which the signatures appear at various offsets within the frame due

{
 "dIP": "192.168.0.2",
 "dPort": 80,
 "data": "R0VUIC9zaXRlcy9kZWZhdWx0L3RoZW1lcy9mcmFtZWR5bmFtaWMv...
 "id": "(' 192.168.0.1', '192.168.0.2', 52217, 80)",
 "trProto": 6,
 "protocol": "HTTP",
 "sIP": " 192.168.0.1",
 "sPort": 52217
},

Fig. 1. Example record in data set.

to preceding protocol headers, which is a natural situation in real network traffic data.

Table 3. Unique signatures in considered application protocols.

Protocol	Unique pattern
HTTP	"GET /"
	"PUT /"
	"POST /"
	"HEAD /"
	"TRACE /"
	"DELETE /"
	"OPTIONS /"
SSH	"SSH-"
SMTP	"220 "
	"220-"

2.3 Classifier in Hardware

The SDM system transfers frames over 512 bit bus. Each frame starts with the headers of low-level protocols such as Ethernet, IPv4 or IPv6, TCP or UDP. As a result, the start of the application payload may appear with certain offsets on the bus, namely 2 bytes from the position 0 or with $2 + 4k$ bytes, where $k = 1 \ldots 16$. Fig. 2 shows that the proposed circuit implementation of the classifier consists of three levels of combinational logic.

In the first level, one coder is connected to each byte of the word (64 coders, in total). There are four types of the coders (c1, c2, c3, c4) because of the 4-byte offsets. Each coder implements a mapping from the set of characters allowed for the given position to a set of 8-bit values in which just 2 bits are not zeros. The mapping functions of the coders are given in Table 4. This remapping implemented by coders allows for a fast signature detection in the subsequent

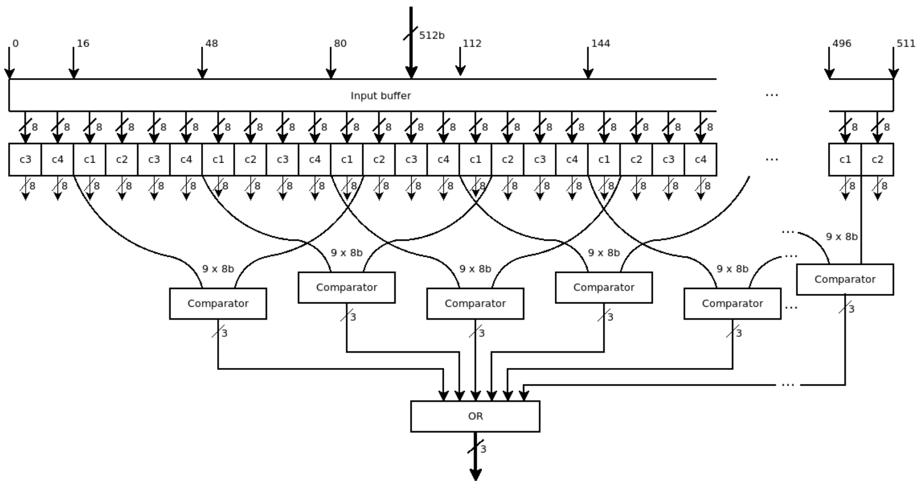

Fig. 2. Proposed classifier as a combinational circuit

level of comparators. All possible occurrences of the application data within the buffer are thus processed in parallel.

The second level consists of comparators. Each of them compares the outputs of nine coders (note that the longest signature contains 9 characters) with the unique patterns identified for the considered application protocols. If a particular application protocol is detected then its 3-bit code is visible at the output of the comparator (001 - HTTP, 010 - SMTP, 100 - SSH, 000 - unknown).

Finally, at the third level, all 3-bit codes are fed to an OR gate which indicates a presence of one of the expected application protocols or unknown protocol (000).

2.4 Circuit Optimization Using CGP

Based on our previous experience, we assumed that parameters of the circuit optimized by a professional FPGA design software can be improved if an evolutionary optimization is employed. As the whole classifier is a relatively complex circuit to be optimized, we propose to optimize its components - the 64 (combinational) coders. Each of the coder types c1, c2, c3 and c4 will be optimized by CGP. We will utilize a standard CGP as defined in [12]. Advanced techniques, such as circuit decomposition [14] or functional level evolution [15] are not needed in this case.

In CGP, a candidate circuit is modeled as a directed acyclic graph and represented in a 2D array of $n_c \times n_r$ processing nodes. Each node is capable of performing one of the functions specified in Γ set. The setting of n_c, n_r and Γ significantly influences the performance of CGP. Current FPGAs utilize 6-input LUTs as building blocks of all circuits. However, employing CGP with 6-input nodes (each of them encoded using 64 bits in the chromosome) would lead to

Table 4. Mapping functions in the coders. The * symbol means: "not utilized in a particular coder"

coder 1	coder 2	coder 3	coder 4	output
space	space	space	space	00000011
/	/	/	/	00000101
T	E	S	T	00000110
E	N	T	D	00001001
O	O	A	E	00001010
G	U	L	C	00001100
P	R	0	I	00010001
H	P	H	-	00010010
D	2	G	R	00010100
2	S	V	*	00011000
R	Y	E	*	00100001
S	A	N	*	00100010
B	C	R	*	00100100
C	T	K	*	00101000
A	L	*	*	00110000
I	*	*	*	01000001
otherwise	otherwise	otherwise	otherwise	00000000

long chromosomes, complex search spaces and so inefficient search procedures. We propose to optimize the coders at the level of 2-input nodes (encoded using up to 4 bits) and let the professional circuit synthesis software implement the resulting optimized circuits using 6-input LUTs in the FPGA.

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 nodes can be used as the inputs for a given gate. The primary inputs and the outputs of the nodes are labeled $0 \ldots n_c \cdot n_r + n_i - 1$ and considered as addresses which links can be connected to. In the chromosome, each node is then encoded using three integers (an address for the first input; an address for the second input; a node function). Finally, for each primary output, the chromosome contains one integer specifying the connection address. Figure 3 shows an example and a corresponding chromosome.

CGP utilizes a search method known as $1 + \lambda$, where λ is the population size [12]. The initial population is randomly generated. 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.

The fitness function consists of two steps. First, the fitness value is determined as the number of bits correctly calculated for all possible assignments to the inputs. If a fully functional circuit is discovered (in the case of 8-input/8-output coders, this fitness value is $8 \cdot 2^8$) then, in the second step, the fitness function is modified because the goal is to minimize the number of gates. The fitness is then

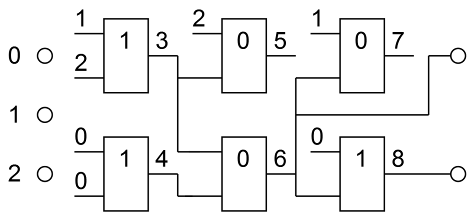

Fig. 3. Example of a 3-input circuit. CGP parameters are as follows: $n_i = 3$, $n_o = 2$, $l = 3$, $n_c = 3$, $n_r = 2$, $\Gamma = \{\text{AND } (0), \text{ OR } (1)\}$. Gates 5 and 7 are not utilized. Chromosome: 1,2,1; 0,0,1; 2,3,0; 3,4,0; 1,6,0; 0,6,1; 6, 8. The last two integers indicate the outputs of the circuit.

reflecting the functionality and the number of used gates. Delay is not explicitly optimized; however, its maximum value is implicitly determined by n_c.

3 Results

The experimental evaluation consists of the following steps: (1) conventional implementation of the proposed classifier; (2) CGP-based optimization of selected subcomponents of the circuit; (3) resynthesis of the classifier with optimized subcompoenents; (4) verification of the quality of classification. As mentioned earlier, we will implement and optimize in the FPGA only the key component of the classifier – the 9 coders with a comparator (9CC circuit, in short).

3.1 Circuit Design and Implementation

The 9CC circuit was behaviorally described in VHDL and synthesized into the Xilinx Virtex-7 XC7VH580T FPGA using Xilinx ISE Project navigator 14.4 tool. The target FPGA contains 6-input LUTs whose latency is 0.043 ns. We set the circuit delay as the main optimization target for the synthesis tool. The resulting circuit contains 188 LUTs and exhibits a delay of 4.621 ns.

3.2 Optimization by CGP

There are 4 types of coders in the classifier circuit; each of them with 8 inputs and 8 outputs (Fig. 2). These coders are optimized by CGP operating at the gate level. CGP is used with the following parameters: $n_i = 8$, $n_o = 8$, $n_c = 50$, $n_r = 12$, $L = n_c$, $\lambda = 4$, $h = 5$. In order to obtain basic statistics, each run consisting of 5 million generations was repeated 10 times. This setting of the parameters was recognized as useful after some experimenting with CGP in this task. Because of the limited space, experimental results are given, as an example of our methodology, only for one parameter – the function set. We compared CGP utilizing all logic functions over two inputs (except logic constants) in the function set (Γ) against and a reduced set (Γ_r which includes in1, and, or, xor, not_in1, not_in2, in1 and not_in2, nand, nor) in the case of coder 1 and 2. Table 5 gives the minimum, maximum and mean number of gates obtained for each coder. One can observe that while the mean values are lower for Γ, the minimum values are lower for Γ_r. Hence Γ_r was used in further experiments.

Table 5. The number of gates obtained by CGP using Γ_r and Γ (see $*$)

	coder 1	coder 2	coder 3	coder 4	coder 1*	coder 2*
Min	58	60	59	39	60	61
Max	83	84	83	56	94	100
Mean	72.25	70.33	69.04	44.81	70.25	69.23

3.3 Classifier Resynthesis Using Optimized Coders

The most compact implementations of coders obtained from CGP were translated to VHDL and utilized in the VHDL code of the whole 9CC circuit of the original components. The modified 9CC circuit was synthesized with the same setting as reported in Section 3.1. The results of synthesis are given in Table 6. One can observe that both crucial circuit parameters were improved. The area was reduced from 188 to 160 LUTs. The original as well as optimized latency is safely within the requested limit of 7 ns. However, the optimized implementation allowed us to increase the spare latency (to $7.0 - 4.156 = 2.844$ ns, see Table 6) which can be used to connect the 9CC circuit with the OR logic and subsequent components of the whole application.

Table 6. Results of synthesis of the 9CC circuit for the Xilinx Virtex-7 XC7VH580T FPGA.

Parameter	Optimized by CGP	Original description
LUTs	160	188
Delay logic [ns]	0.344	0.430
Delay net [ns]	3.812	4.191
Delay [ns]	4.156	4.621
LUT levels	8	10

3.4 Quality of Classification

The quality of classification was evaluated offline, utilizing a software model that we have developed for the proposed classifier. The evaluation was performed using both data sets, but we considered two scenarios for classification: (1) thinned traces containing first payload packets only and (2) complete traces containing all packets (i.e. the classifier's objective was to analyze every incoming packet in this case). The output of the proposed classifier was verified against the L7 filter which provides 100% correct results for considered protocols. We calculated Precision and Recall metrics:

$$Precision = \frac{TruePositve}{TruePositve + FalsePositive} \tag{1}$$

$$Recall = \frac{TruePositve}{TruePositve + FalseNegative} \tag{2}$$

Precision informs us how many packets assigned to a given class are really correctly assigned. Fig. 4 shows that HTTP, whose representation is rich in our data sets, is classified perfectly. The reason for lower percentages of Precision in the case of other protocols is the fact that their unique patterns are relatively short and can easily appear inside of other protocol packets. As the subsequent packet processing is done in software precisely the incorrectly classified protocols will be recognized anyway. The software task is simpler than that of the original one. Software must only verify the labelled traffic and dismiss false positives.

Considering the whole SDM, which the proposed classifier is targeted for, the Recall is even a more important metrics. High Recall values indicate that if a given application protocol is present in the traffic data, it is detected with almost 100% probability and thus no information is lost.

Fig. 4 does not give any data for SSH in CESACO (first packet) data set. The reason is that there is no record with SSH in this data set.

4 Conclusions

In this paper we presented a new application of the evolutionary design and optimization – the optimization of circuit implementation of the application protocol classifier intended for high speed networks. The proposed solution exploited the fact that results of conventional FPGA synthesis tools can further be improved when selected circuit components are optimized by CGP and the optimized versions replace the original ones in the target circuit. The whole classifier is composed of 9CC circuits working in parallel. By optimizing just the 9CC circuit we obtained a very good estimate of the total area of the classifier (16 instances of 9CC will be needed) and the total delay (which is the delay of 9CC plus a small delay of the OR network as seen in Fig 2).

The proposed solution is capable of fast detection of key application protocols using a single packet only. It exhibits excellent Recall values (no monitored application protocols are missed). It is planned that further and detailed packet processing, which can improve the precision parameter of the hardware classifier, will be handled in software by the SDM framework. Our future work will consist in including other application protocols into the proposed hardware solution.

Acknowledgments. This work was supported by the Czech science foundation project 14-04197S, Brno University of Technology project FIT-S-14-2297 and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

References

1. Karagiannis, T., Broido, A., Brownlee, N., Claffy, K., Faloutsos, M.: Is P2P dying or just hiding? In: Global Internet and Next Generation Networks, Dallas, Texas, Globecom 2004 (Dec 2004)
2. Karagiannis, T., Papagiannaki, K., Faloutsos, M.: Blinc: Multilevel traffic classification in the dark. SIGCOMM Comput. Commun. Rev. **35**(4) (2005) 229–240
3. Yoon, S.H., Park, J.W., Park, J.S., Oh, Y.S., Kim, M.S.: Internet application traffic classification using fixed ip-port. In: APNOMS. Volume 5787 of Lecture Notes in Computer Science., Springer (2009) 21–30
4. 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) 50–60
5. 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) 512–521
6. Tongaonkar, A., Keralapura, R., Nucci, A.: Challenges in network application identification. In: Presented as part of the 5th USENIX Workshop on Large-Scale Exploits and Emergent Threats, Berkeley, CA, USENIX (2012)
7. Bernaille, L., Teixeira, R., Salamatian, K.: Early application identification. In: Proceedings of the 2006 ACM CoNEXT Conference, New York, NY, USA, ACM (2006) 6:1–6:12
8. Zilberman, N., Audzevich, Y., Covington, G., Moore, A.: Netfpga sume: Toward 100 gbps as research commodity. Micro, IEEE **34**(5) (2014) 32–41
9. Friedl, S., Pus, V., Matousek, J., Spinler, M.: Designing a card for 100 gb/s network monitoring. Technical report, CESNET (2013)
10. Park, B.C., Won, Y., Kim, M.S., Hong, J.: Towards automated application signature generation for traffic identification. In: Network Operations and Management Symposium, 2008. NOMS 2008. IEEE. (April 2008) 160–167
11. 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) 1725–1733
12. Miller, J.F.: Cartesian Genetic Programming. Springer-Verlag (2011)
13. L7filter: Application layer packet classifier for linux (l7-filter) (2009)
14. Stomeo, E., Kalganova, T., Lambert, C.: Generalized disjunction decomposition for evolvable hardware. IEEE Transaction Systems, Man and Cybernetics, Part B **36**(5) (2006) 1024–1043
15. Shanthi, A.P., Parthasarathi, R.: Practical and scalable evolution of digital circuits. Applied Soft Computing **9**(2) (2009) 618–624

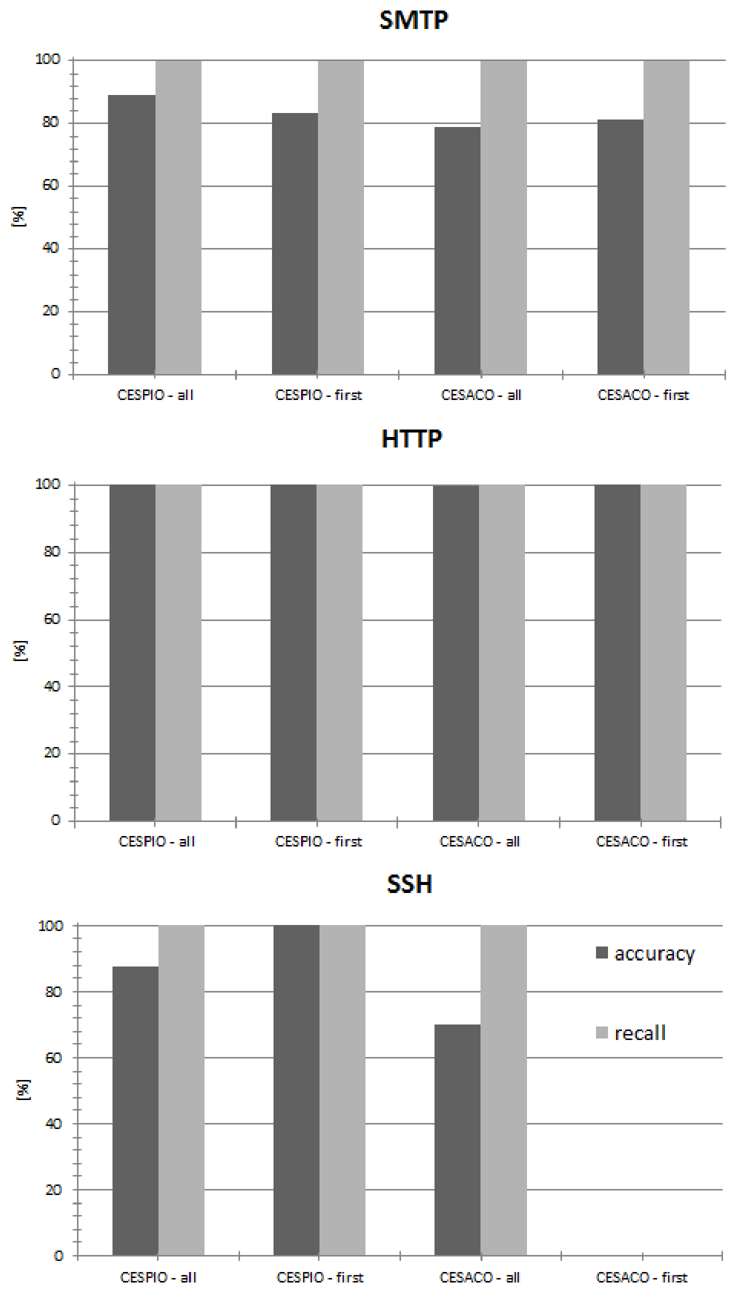

Fig. 4. Precision and Recall percentages for three classified protocols. Test data consists of first packets only (first) and complete network records (all).