Open-source flexible packet parser for high data rate agile network probe

Franck Cornevaux-Juignet, Matthieu Arzel, Pierre-Henri Horrein, Tristan Groleat, Christian Person

Open-source flexible packet parser for high data rate agile network probe

Franck Cornevaux-Juignet*, Matthieu Arzel*, Pierre-Henri Horrein*, Tristan Groléat† and Christian Person*

*IMT Atlantique Bretagne-Pays de la Loire campus de Brest
Technople Brest-Iroise CS 83818 29238 Brest Cedex 3 FRANCE
Email: firstname.lastname@imt-atlantique.fr

†OVH
230 rue Roland Garros, 29490 Guipavas FRANCE
Email: tristan.groleat@corp.ovh.com

***Abstract*—The development of a network centered life has increased overall data rates in core networks. Thus, data centers face the challenge to provide always more services at higher data rates while reacting quickly to complex failures and more powerful attacks thanks to efficient network forensics. Moreover, Software-Defined Networking (SDN) becomes a standard which offers agility but also requires forensic devices able to handle multiple configurations.**

Although conventional software probes are programmable and thus agile, they cannot support high data rate packet processing any more. Probes could benefit from Application Specific Integrated Circuits (ASIC) to cope with high data rates, but ASICs development time of many months makes them unable to satisfy agility requirements. With reconfiguration ability and high throughput processing without packet loss, Field Programmable Gate Arrays (FPGA) are the key technology chosen by some companies, such as Microsoft, Amazon and OVH, to be integrated into smart Network Interface Cards (NIC). Nevertheless, while high performance criteria is fulfilled, current FPGA probes benefit from an agility still limited to their conventional firmware upgrades which require proprietary tools and hardware-design time and knowledge.

This paper proposes the first solution to offer FPGA probes with runtime agility thanks to a flexible packet parser which can be parameterized continuously by a software, endorsing complex tasks and SDN control. This allows a live adaptation of protocol processings from computer host alongside handling packets at line rate without data loss. The proposed parser is open-source and easily usable by network engineers through a Python software API. Benchmark results illustrate the performance of the agile high-level probe implemented on a NetFPGA SUME board, with XC7VX690T FPGA. 60 millions of 64-byte packets are counted based on features provided at runtime. These are selected by the software part, allowing the detection of different volumetric attacks within a few tens of microseconds. This represents a 40 Gb/s traffic of smallest Ethernet packets with no packet loss. With adequate boards, the generic design of the probe offers 160 Gb/s data rates and beyond on modern hardware, assuring probe scalability.

I. Introduction

In September 2016, the web service provider OVH has been one of the victims of an attack never encountered before [1]. A botnet infected up to 150,000 poorly protected security cameras to generate up to 1 Tb/s of malicious traffic. In October 2016, a similar attack targeted the Dyn company, provider of many well-known online services [2]. Both attacks were the results of the Mirai botnet, controlling enough connected objects to create overwhelming traffic. This is an example of using Internet of Things devices as a leverage [3]. The development of the Internet has led to the emergence of numerous services and an increase of bandwidth to end users. The current expansion of connected objects and computer equipment has multiplied the number of possible sources of threats.

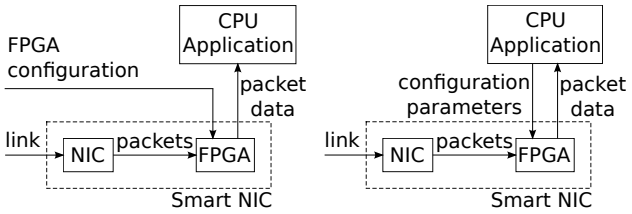
Network actors monitor the traffic to detect anomalies in traffic and to counter potential threats or malfunctions. This live forensics must be done without packet loss and with the smallest latency in order to react as fast as possible on high speed links. Moreover, traffic monitoring systems must be compliant with SDN agility [4]. This paradigm dissociates the data plane and the control plane of the infrastructure to set a level of programmability and follow the evolution of the traffic. The plane separation unties the control from network equipment leading to the network architecture abstraction and a global management adapting the infrastructure to the traffic content. Therefore, packets must be processed at line rate and in a flexible way to allow the infrastructure agility. Conventional probe solutions are not adapted to answer this challenge of an agile monitoring at high data rates of 40 Gb/s or 100 Gb/s links, and even 400 Gb/s in data centers.

Despite their processing flexibility, software based probes lack processing power to concurrently sustain high data rate traffic and normal processings [5]. To improve processing capabilities of network equipment, common solutions use specialized hardware implemented as ASICs. Despite the performance, ASIC system development time and production cost are prohibitive for processing flexibility [6]. The current considered solutions are smart NICs composed of an ASIC part for NIC and an FPGA to offload CPU processings, as seen in Microsoft Azure [7]. This conventional architecture is shown in figure 1(a). FPGAs offer a good compromise between high data rate processing and flexibility. The programming paradigm is similar to ASIC, meaning that FPGA data pipelines can achieve high data rate processing through high parallelization, well-suited to network processing. Yet, unlike ASICs, FPGAs are not designed for a unique task: they can be programmed with a new design without the need to produce a new chip. This increased agility makes them better candidates for SDN. However, this compromise is not perfect. Development time for FPGA is high, and requires expensive tools, which counterbalances this flexibility. FPGA agility is usually limited to periodic firmware updates, instead of runtime modifications to adapt the processing to the incoming traffic.

While a solution would be to integrate all expected processings in the design loaded in the FPGA, and then dynamically select the requested operation, this is not possible due to limited FPGA resources. Only an adapted and restricted probe configuration can be run at a given time. Using a set of pre-designed firmwares, selected at runtime according to the situation, the FPGA can be reprogrammed within a few

This research is supported by Brittany region, Finistère regional council, Brest Métropole funds.

978-1-5386-0683-4/17/$31.00 2017 IEEE



(a) Conventional smart NIC (b) Proposed smart NIC

Fig. 1. Conventional and proposed smart NIC architectures

seconds. However, during these few seconds, the probe cannot monitor the traffic. If no pre-designed firmware is adapted to a new situation, a new FPGA configuration has to be designed by a specialist having knowledge in hardware and network protocols. This work is complex and time-consuming, even with the help of high-level tools, to meet the required performance in terms of latency and throughput while fitting FPGA constraints. Discarding the design time, firmware generation is a matter of hours, even days for complex architectures. Obviously, common FPGA-based solutions provide a coarse-grain agility but they do not allow re-configuring probes to continuously adapt the monitoring to the incoming traffic.

This paper presents an innovative FPGA-based flexible packet parser to support a high performance adaptative network monitoring probe for live forensics aware of traffic content. The proposed FPGA part is designed to be runtime reconfigurable and to process packets at line rate, using a generic parser design. The software part runs forensic applications and controls the runtime FPGA configuration. This architecture is shown in the figure 1(b). Thus, network engineers avoid FPGA development and are just using a dedicated API, able to provide only useful data according to dynamically evolving application needs, without loss. A high performance parser must face multiple challenges [8]:

- line-rate throughput,
- headers sequential dependency,
- protocols and headers heterogeneity,
- header format programmability.

The proposed architecture takes advantage of the combination of hardware processing power and software flexibility to answer these challenges. It relies on a static parser processing packets at high data rates, which is made configurable thanks to parameters sent from the software application. This allows a runtime dynamic configuration of processed protocols without shutting down the probe, which is not possible in current solutions. Moreover, once the FPGA is configured, specialized proprietary tools are not needed to configure the parser. A software API ensures an efficient software integration and an easy adoption by end-users.

The paper is organized as follows. Section II introduces several works in the area. Section III describes the flexible probe and parser implementation details. Section IV presents obtained results and Section V concludes with future development.

II. Related work

A. Software probes

The development of a software based probe is equivalent to the development of programs. Thanks to CPU flexibility, software based probes are very flexible and can perform any processing. The wide variety of existing development tools allows applications to be portable on any software platform.

However, these solutions have performance issues in terms of data rate. Moreno *et al.* [5] showed that there is a boundary on smallest 64-byte packets processing. Despite fully optimized drivers, software tends toward the loss of packets when approaching 10 Gb/s, even with basic processing such as packets counting. This analysis is shared by Gallenmüller *et al.* [9]. The CPU takes at least 100 cycles to receive and transmit a packet with the most efficient driver. The analysis of a simple processing, packet forwarding, with a lookup table of 256 MB in size leads to 7.1 Mpps processed. This is less than 14,88 Mpps of 64-byte packets that need to be processed in a worst case scenario on a 10 Gb/s link.

For the last 20 years, networking devices have improved datapath bandwidth more than CPU devices [4]. This evolution has led to the emergence of other solutions than software based solutions. Despite a longer time to market because of longer development and verification, network devices are always ahead in terms of bandwidth.

B. Hardware probes

The extreme parallelization of FPGA design allows to pipeline packets processing and to reach high data rates. Xilinx with Ultrascale+ [10] and Intel with Stratix 10 [11] announce devices supporting 400 Gb/s data rate.

An example of high data rate application was described by Groléat et al. [12]. They proposed a standalone SVM classifier labeling a packet depending on the flow of the packet. The flow is detected with the classical 5-tuple IP source, IP destination, port source, port destination and protocol. The implemented design supports 10 Gb/s of 64-byte packets on an FPGA board.

FPGA implementation of network probes has attracted some interest in recent years. Antichi *et al.*[13] developed a solution where the hardware part of the probe unloads CPU with the packet processing. An onboard memory allows to share sketches between hardware and software. An anomaly is detected based on aggregate data of sketches. The implementation was done on a NetFPGA 1G card for a full 1 Gb/s link. Between full software and part hardware probes, the decrease in CPU usage from 100 % to *quasi* 0 % comes with a reduction of packet drop rate. However, the hardware part must be reconfigured to process other types of traffic. Garnica *et al.* [14] designed a 10 Gb/s hardware-software probe for URL legal filtering. This throughput is maintained with 64-byte packets. The hardware part extracts IP addresses and filters depending on local rules. The software part holds the URL and IP correspondence to decide which URL filter has to be applied. The implementation is done on a NetCope server. The results are extrapolated as functioning at 100 Gb/s on Virtex7 FPGA.

However, a common characteristic of all these solutions is a fixed packet parser. Data extracted from packets and processings are only adapted to the current solution. Modifications in incoming traffic could invalidate processings. Even despite the use of a software part, processings are limited to subsets of packets. FPGA development reaches its limits when the design must be modified. Indeed, the update of a design requires the modification of the design and the reprogrammation of the chip. Time is needed for modifications and functioning tests, besides specific competences not widespread in network community are required. Moreover, the reconfiguration leaves the probe unable to process incoming packets during reconfiguration time.

C. Configurable solutions

To counter drawbacks tied to hardware development, some solutions have been proposed combining an hardware part for basic operations on a huge workload, and a software part for smarter processing on a reduced dataset, and for the configuration of the hardware design.

A NoC-Enhanced FPGA packet parser is proposed by Bitar *et al.* [15]. A NoC hosts a header parser on each node and can reconfigure routes between parsers. Based on synthesis results, they extrapolate a design working at 400 Gb/s with 512-byte packets. The ultimate purpose of this design is to hold dynamic protocol configuration with a partial reconfiguration of router nodes. Hager *et al.* [16] designed a probe which integrates a reconfigurable core processing in a NetFPGA 10G board design. Processing is set to an optimized routing table to forward packets. However, the header parser extracts only destination IPv4 addresses, which reduces greatly the interest of partial reconfiguration.

Pus *et al.* [17] presented a pipelined design of packet header parsers reaching data rates of 100 Gb/s. This parser architecture supports an automatic high level parser generation tool [18]. Using the P4 language allows a high level description of header parsers. The presented P4-to-VHDL parser generator uses this description to create an HDL design. The resulting architecture is parsing packets, as the original, at 100 Gb/s after synthesis. This tool allows faster design of data extraction from packets, but requires a full configuration of the FPGA when new protocols are considered, stopping running processing.

In the same field of software assisted development, HLS languages such as G [19] and PP [20] were introduced for packet parsing. A specific architecture is used as a result of the HLS language compilation assuring high data rates, up to 400 Gb/s. Parser stages are based on microcode allowing header parsing modifications. If too significant changes are done between two updates, the parser architecture must be regenerated. Otherwise, the design is globally regenerated and reconfigured. With the PX language [21], deployment flexibility is ensured by the tool flow which checks if an update of the current implementation firmware is possible before generating a new bitstream. A generated OpenFlow packet classifier processes packets at a data rate of 100 Gb/s. The outcome of this whole research is the SDNet full product [22] allowing the generation of a dynamic solution thanks to partial reconfiguration of the FPGA at 100 Gb/s. Bringing an easy access to FPGA systems for network specialist is necessary for the development of high performance processings. However, presented solutions need to be reconfigured at some point. During the reconfiguration time, even with partial reconfiguration, the parser and the probe are unable to process packets. Moreover, design flexibility relies on external tools, which are proprietary, inducing design portability issues between FPGA's vendors.

III. Flexible network probe

A. Hardware/software system

In current utilization of smart NICs, the FPGA is seen as a hardware accelerator which can be upgraded. Even though the CPU offload allows to reach high performance monitoring, FPGA processings are stopped during the reconfiguration time. Processing adaptations are then limited to sparse firmware upgrades. This low flexibility meets the requirements of SDN, but is not enough to be adapted to incoming traffic.

To overcome this limitation, this paper proposes a hybrid architecture between the FPGA of the smart NIC and the CPU to support runtime adaptations and live adaptation to traffic. The static configurable design on the FPGA offers a live configuration *via* parameter settings, as well as CPU offloading. The software part runs monitoring applications and handles the settings control of the FPGA. The configuration possibility creates a feedback loop between the CPU and the FPGA allowing to refine hardware processings when needed. The CPU can then work on a chosen fraction of data produced by the FPGA, such as specifically filtered packets or metadata like packet counters. This idea takes advantage of both software flexibility and hardware raw performance to tackle high performance traffic monitoring.

1) Network users friendly: A static design with parameters permits the abstraction of the probe for the user. Indeed, no hardware reconfiguration of the probe is necessary, avoiding the use of external specialized proprietary tools for FPGA development. Configuration parameters are figured out in software, where an API allows an easy interaction with them. Network applications can then interact with preprocessing functions in a transparent manner.

2) High throughput: The FPGA part of the design contains a static architecture set to support high data rates. The global probe is then reliable and resilient to saturation traffic. The traffic is fully processed by the probe without packet loss. Data forwarded to the software part are precise and chosen by the operator, not by packet sub-sampling.

3) Runtime adaptative processing: The configuration of the FPGA design is performed through parameters provided by the software application. These parameters modify the functionality of preprocessing units designed on purpose. This offers flexibility of the probe, without the need for time-expensive FPGA reconfiguration. Configuration is then possible at runtime allowing the probe to be content-aware and to adapt itself to the incoming traffic. Moreover, the hardware part runs processings with a controlled low latency. The feedback loop is then low latency, enabling a reactive probe.

B. Parameterized Packet Parsing

1) Flexible design: This part presents a packet parser consistent with the paradigm presented previously. Packet parsing is the required first step responsible for detecting and extracting required features in a packet, often in protocol headers. A packet parser supplies adequate inputs from packets to next operations in the processing chain. The most commonly used features are source and destination IP addresses, source and destination UDP or TCP ports, and the transport protocol. These form the classical 5-tuple for network analysis by tracking flow information from this tuple. However, algorithms could need complementary information on packets like layer 7 protocol header fields or non common protocol layer specific to a type of application server. To provide such diversity in information, a static parser can not be considered. Packet parsing is a per packet operation on the datapath, justifying its implementation on FPGAs at high data rate.

Using configuration parameters allows a hardware design to gain flexibility. The one wanted in this case is on the type of protocol header to process. A packet is a succession of bits organized in several headers and payload data. A specific feature in a packet needs 2 pieces of information to be extracted, the header location and its location inside the

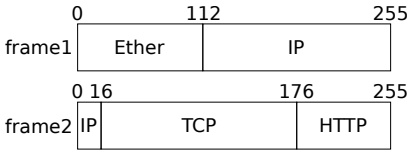


Fig. 2. Packet frames example with 64-byte packet and 256-bit datapath width

header. The protocol encapsulation in a packet lets a lower-level header to contain information on the next header, the service access point (SAP). However, a protocol can have a variable size. In this case, the size information is contained in the current header. A header feature, a field or aggregate of multiple fields, is at a fixed offset from the beginning of the header and has a fixed size. To extract a feature from a packet, the parameters needed are :

- a header size or the location of the corresponding size field,
- the location of the SAP field in the header to know the next header,
- the location of the feature inside the corresponding header.

As an example, to get the TCP destination port out of a TCP/IPv4 packet, the offset of the field from the beginning of the header and the end offset of the field are needed. The global TCP header must be extracted from the IPv4 packet, requiring the knowledge of the location of IPv4 length and protocol fields. Obviously, the IPv4 packet has to be beforehand extracted from the Ethernet frame.

2) Global architecture: When processed in the FPGA, the packet is not continuous, it is composed of frames, whose number depends on the datapath width. Frame width on the FPGA is chosen alongside an operating frequency to match a target link speed. The extraction of a feature from the packet is conditioned by its location in each frame. A feature is determined in the header where it is needed to be extracted, it is therefore necessary to locate the corresponding header. The figure 2 shows the composition of the 2 frames of a 64-byte Ethernet packet on a 256-bit wide datapath. It is easy to notice that a header can be broken into 2 frames, as well as features of interest.

The parser must then locate the header in each frame to locate and extract wanted features. Moreover, the built parser must extract any feature from any header. This genericity infers the separation of the packet parser between header parsing and feature analysis. Indeed, the number of features to extract by protocol is not known *a priori*. This separation allows in addition to share feature analyzers between all headers.

Due to data encapsulation in network packets, protocol headers need to be processed in encapsulation order. It is therefore not possible to handle all headers of the packet at the same time. The figure 3 shows the packet parser full architecture. High data rates are achieved by analyzing headers in pipelined stages. Each stage is analyzing one layer of encapsulation. On the other hand, feature extraction does not dependent on information inside packets, but can be determined with fixed offsets from header beginning. As a result it is possible to make it parallel. Header data for each header are produced at different times due to the location in the pipeline. Introducing a suitable delay after each header location synchronizes header data with the corresponding packet data.

The parser configuration is handled by parameters for the location and the extraction. Each protocol is uniquely identified during the parsing process. This lets configured analyzers to detect the right protocol to work on.

3) Header parsing: This module extracts the current protocol location information in frames and determines the following header if possible. The location is done with the size of the header or the size field inside the header to locate it in packet frames. This location determines min and max indices of the header in the current frame if needed. Protocol decapsulation uses the SAP field to determine the next header in the packet.

The figure 4 presents header parsing architecture. The first step in the module fetches the descriptor corresponding to the type given by the previous block. This descriptor is the configuration parameter which contains information about the analysis of the current header : the header global id, the length field size and offset, the protocol field size and offset. For the length and protocol, the offset and size refer to the offset and the size of the corresponding field in the header. They are extracted as any feature as shown later in the paper. Thanks to the length field, the current header can be located inside packet frames. The location consists of determining the header min index and max index in each packet data frame. The protocol field value is combined with the current protocol global id in order to find the next header protocol global id. This protocol is processed by the next block in the pipeline. Referring to the example in III-B1, such parameters to decapsulate IPv4 encapsulation and get TCP header are :

- IPv4 length field offset : 4 bits
- IPv4 length field length : 4 bits
- IPv4 protocol field offset : 72 bits
- IPv4 protocol field length : 8 bits
- TCP protocol field value : 6

4) Feature analysis: This module extracts from one header several features whose number is selected at synthesis time of the design. Features are located with minimum and maximum offsets from the beginning of the header. With the module configuration, the type of header to use can be selected as well as min and max offsets of features to be extracted. On the opposite, the feature max width is selected at synthesis time. To extract a feature, min and max indices of the feature in each frame must be known. These indices are computed from header location indices and feature offset in the header.

The figure 5 presents the different steps leading to features extraction. The analyzer receives all header data coming from all the header location modules and select the proper header depending on the configured header type. When indices of the feature in the frame are calculated, the feature part is extracted from the frame and set in proper indices in a register, as shown in figure 6 for the IP destination address. When all the parts of the feature are extracted, the feature is considered extracted. The packet last frame validates extracted features. To get, for example, TCP destination port field, the begin offset, 16 bits, and the end offset, 31 bits, of the field are needed.

5) Architecture flexibility: The full processing architecture is based on abstract parameters. These parameters are designed to correspond to information needed for data decapsulation. The architecture can extract any feature from any header informed in parameter memory. Parameter values can be defined from the software part, making the packet parser flexible up to a limit due to hardware. Indeed, the hardware resources are not infinite on an FPGA. At design time, a maximum depth of encapsulation has to be fixed. Beyond this depth, the remaining

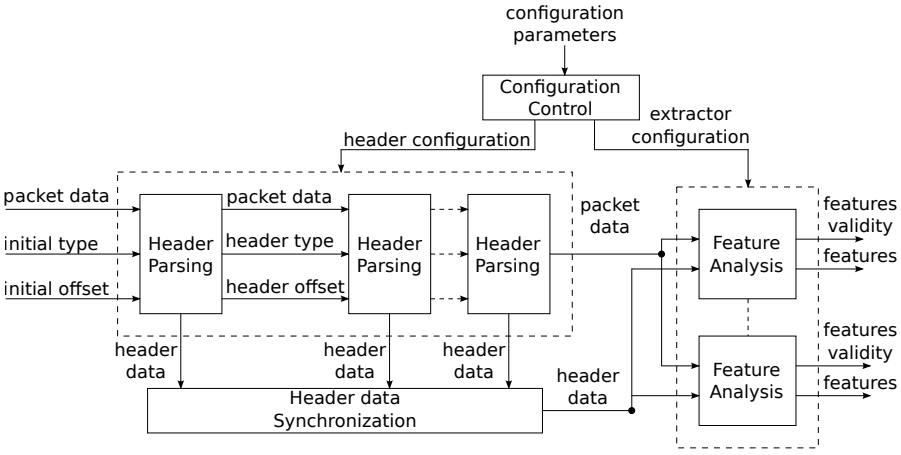


Fig. 3. Packet parser global architecture

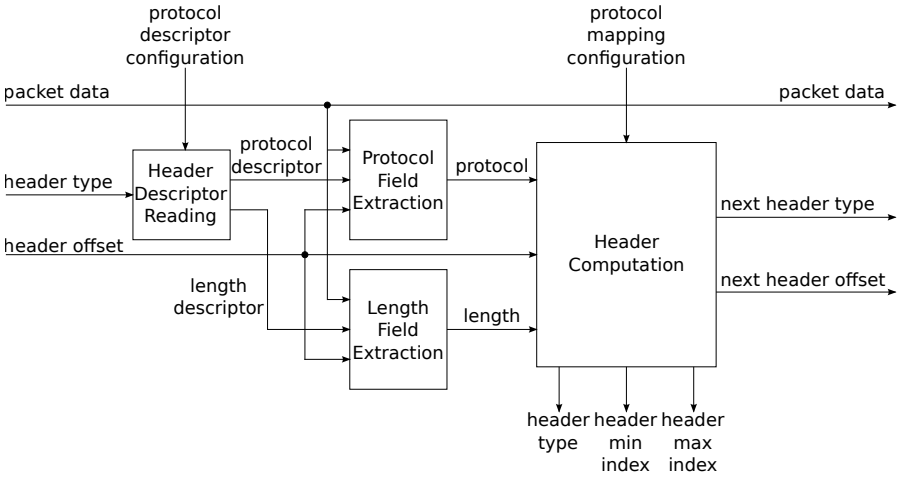


Fig. 4. Packet parser header parsing

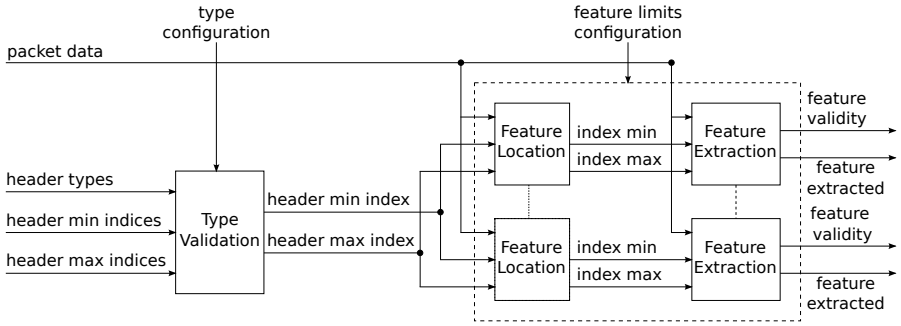


Fig. 5. Packet parser feature analysis

headers cannot be analyzed. Therefore, this depth choice is crucial to insure a complete packet parsing at hardware level. Specific packets with more headers than foreseen are detected and transferred to the software part for further processing.

On the opposite of cutting edge parsing solutions, this architecture is configurable at runtime without stopping probe processing, thanks to simple register one-cycle read and write. Since supported data rate is only determined by data width and clock on the FPGA, the flexible packet parser can be programmed to support high data rates alongside being highly flexible allowing a probe similar to the one presented in the next section.

C. NetFPGA based probe

1) NetFPGA system: NetFPGA is an affordable platform designed for research in high performance network field. This project brings the possibility to design working prototypes of smart NICs. The genericity of the proposed architecture

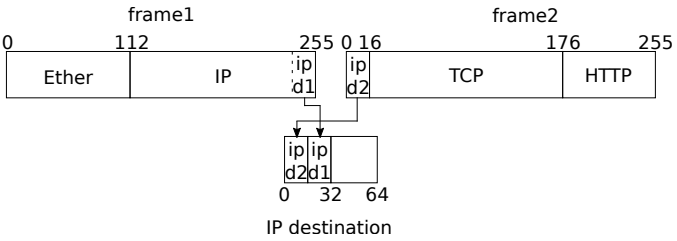


Fig. 6. IP destination address extraction from frames

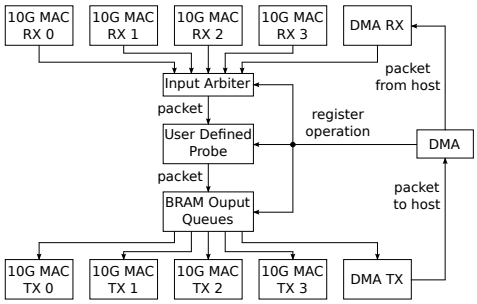


Fig. 7. NetFPGA flow pipeline

allows to work on any FPGA, and so it can be integrated in a NetFPGA system.

NetFPGA platform is designed to sustain high data rate with a Virtex 7 XC7VX690T FPGA part and gains flexibility through communication with a software part. Indeed, the board is primarily designed for 40 Gb/s of traffic with four SFP+ interfaces but an FMC extension card can add interfaces up to 120 Gb/s [23]. The board contains a PCI express connection giving the possibility to communicate with a host.

The project proposes a reference architecture to facilitate integration of user designed modules for 40 Gb/s system. A flow pipeline model is used with external and host forged packets concentrated by the input arbiter in a common flow for all user defined processing, as shown in figure 7. Then, packets are dispatched to different outputs and to host by output queues. A DMA system allows the communication with the host computer through PCI express connection. This implements the separation between register operation, namely reading and writing, and packet transmission.

2) Architecture implementation: The flexibility of the proposed architecture simplifies its integration inside the NetFPGA project. Indeed, the insertion in the pipeline flow requires only the adaptation of communication interfaces. Register operations are used to transfer the configuration parameters to the design. Metadata can be sent to the host through local forged packets. User defined protocols can be used to differentiate metadata sent to software programs. The software part needs the adaptation of calls to NetFPGA driver. This ability of interface adaptation makes the design easily portable to other platforms.

IV. Experimental results

A. Experimental probe

The proposed architecture for flexible packet parser has been implemented and deployed on the NetFPGA smart NIC. The focus of this study is on the parser, which is the primary block for any network processing application. The proposed

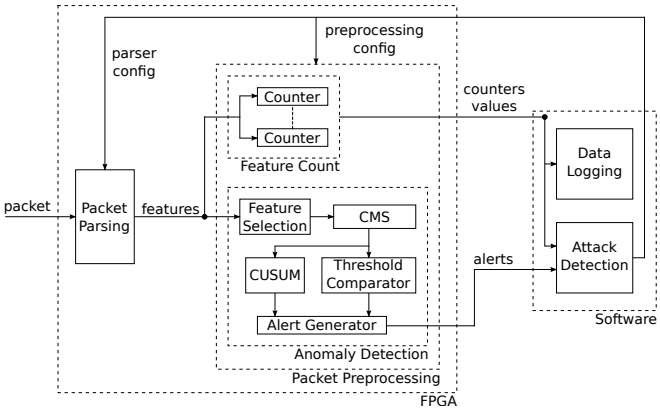


Fig. 8. Test solution architecture

experimental setup aims at providing sufficient proof that the packet parser is able to perform at the expected performance level, and to evaluate its impact on applications in terms of configuration latency, resource usage, and ease of use.

The parser has been coupled to a basic preprocessing unit, in order to form a simple configurable hardware probe as illustrated in figure 8. Required data is requested from a software application, which performs actual processing and adapt the configuration.

The preprocessing unit is composed of filtered counters on selected features and an anomaly detection based on destination IP count. One parametrable filter and one counter are available for each possible extracted feature. The anomaly detection is done with a change point detection over CMS sketches [24] coupled with threshold detection. Counters and alerts generated are transferred to the host machine via PCIe connection for further processing.

Based on this simple probe implementation, a first remark can be made: making use of features from hardware is as easy as processing a FIFO. This means that common data processing elements can be easily integrated, removing the complexity induced by data extraction and frame management. Using a generic unit, the same preprocessing can also be applied to different field without any hardware change. For example, with a counter able to process up to 6 bytes fields, the same architecture can be used to count Ethernet packets or IP packets.

Decisions for fields to monitor and required data is made by software. This is done through a simple software API, which completely abstracts the hardware implementation. The use of hardware for preprocessing is completely hidden from the developer.

B. Benchmark scenario

Wide presence of connected objects facilitates the creation of intense volumetric attacks on any protocol [3]. This malicious traffic aims to exhaust server resources with the reception of a massive number of packets. The overload orientation of these attacks makes them detectable if enough processing power is available. This application is a good candidate to validate the probe: it saturates data links, and it requires a good agility to adapt to the protocol being targeted by the attack. It is also a well-known attack, with well-known countermeasures. The ability to efficiently protect a network from such an attack

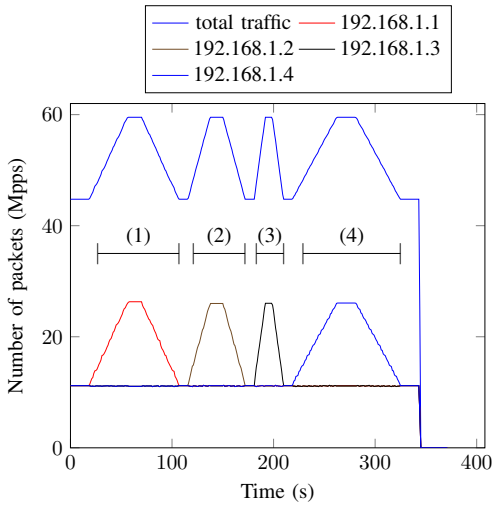


Fig. 9. Incoming packet counters

relies only on the ability to monitor all affected protocols at a sufficient speed.

The proposed test probe is fed with a synthetic traffic composed of a base traffic mixed with attacks on different protocols. Considered attacks are common ones like ICMP ping flood, TCP syn flood, DNS QUERY flood and HTTP GET flood [25]. This list only affects the software part: any attack which can be detected by counting occurrences of a field and comparing to a threshold can be processed by the probe, as long as a configuration is provided.

For the sake of clarity, each attack is targeting a distinct IP address. The base traffic creates a floor of 30 Gb/s of minimal size 64-byte packets, being 44,642 Mpps. Attacks are sent successively completing the base traffic to reach 40 Gb/s, 59,523 Mpps, the maximum rate achievable with the current card interfaces. Global traffic shape received by the probe is visible on figure 9. Different spikes in traffic correspond to different attacks :

- ICMP ping flood on 192.168.1.1 (1)
- TCP syn flood on 192.168.1.2 (2)
- DNS QUERY flood on 192.168.1.3 (3)
- HTTP GET flood on 192.168.1.4 (4)

The generation of this traffic is done by a custom generator with the possibility of producing a 40 Gb/s traffic of 64-byte packets. The testbed is available under an open-source license on NetFPGA SUME for verification [26].

A program developed on purpose in Python is monitoring this incoming traffic. This very basic program detects volumetric attacks, including the ones inserted inside the test, on a destination IP address. This program decides which kind of protocol is monitored. As many features as decided at design time can be simultaneously analyzed. When an alert is set, parameters of the probe are incrementally computed by the Python program, to refine the detection with adapted counter values while keeping track of other possible threats. Decision steps are shown in figure 10. Each step leads to a parameters set for the design coherent with the observed traffic.

This procedure takes advantage of the flexibility of the approach, to selectively monitor protocols according to the traffic shape at a given time. Any protocol can be described using the proposed set of parameters, and software integration

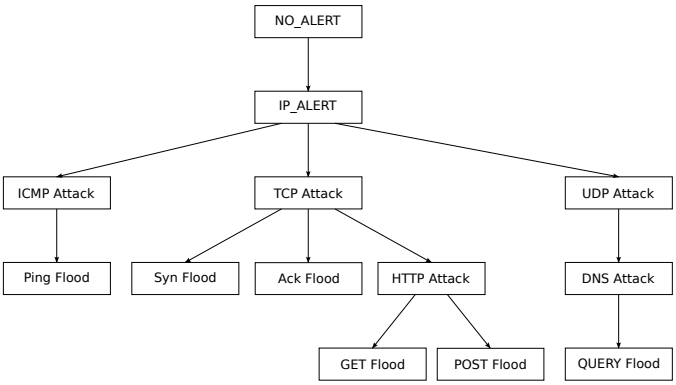


Fig. 10. Detection decision steps

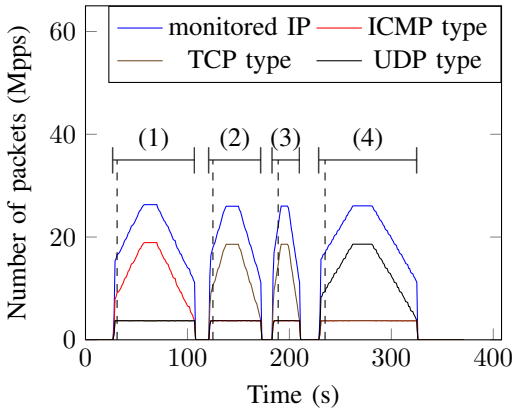
is as simple as calling the corresponding functions. This application is a proof of concept, and only implements basic functionality. More complex forensics could obviously take full advantage of the proposed agile and high data rate probe.

C. Test results and resource usage

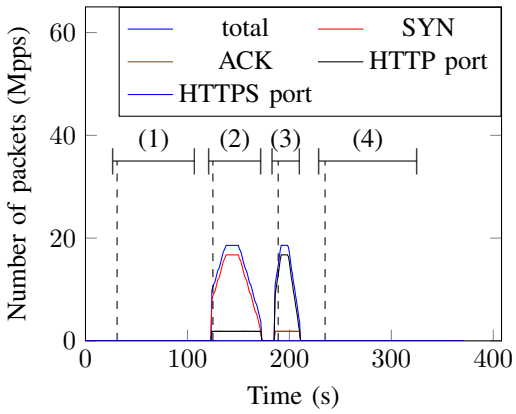
Figure 11 summarizes counter values over time for traffic content-aware specialization on IP, TCP and HTTP protocols, as response forensics for received traffic shown in figure 9. Figure 11(a) shows the recorded traffic for a specific IP, figure 11(b) a refinement on TCP traffic, and figure 11(c) a refinement on HTTP traffic. Holes in the curves represent time intervals when no anomaly is detected, and then no recording is necessary for these protocols. The same behavior can be observed with ICMP, UDP and DNS test traffic. All configuration refinements are done at runtime on demand of the software. Dash vertical lines represent detected attacks. These detections are performed at the beginning of each traffic spike, showing the short reaction time of the probe. Counter values export interval is configurable in a range from microseconds to seconds, directly influencing detection time. The probe is able to differentiate protocols at a rate of 40 Gb/s.

This example shows the possibility to monitor high data rate traffic in an agile way. To monitor the same type of traffic, a classic FPGA approach will reserve resources for each protocol, even if the usage is not optimal. Finally, on a classic approach, if a new protocol needs to be added to the monitoring list, a new binary needs to be generated and the FPGA needs to be reconfigured. For the proposed design, adding simply corresponding protocol parameters to settings is enough and done by the software.

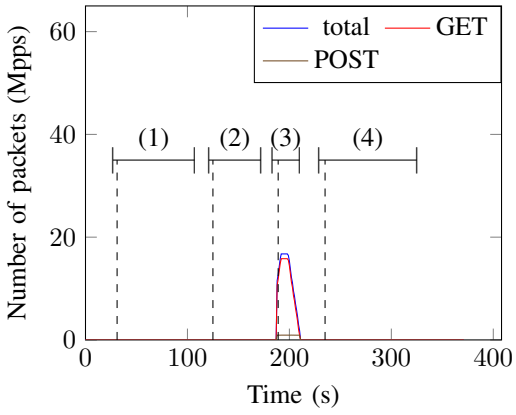
Configuration parameters bring flexibility but require an underlying design which is non protocol-specialized. This could lead to resources over-utilization by the design. For complex traffic, this drawback can be mitigated by the possibility to share resources between protocols. The figure 12 studies the impact of different configurations of the packet parser on resource consumption on the FPGA. Based values used, when not variating, are 5 header parsers, 10 feature analyzers and 128-bit features. These results correspond to a complete implementation process with Vivado 2014.4 tool without NetFPGA interfaces overheads on Virtex 7 XC7VX690T. Feature width amplitude is fixed to extract features from common 32-bit IPv4 address field to 128-bit IPv6 address field. The widest design uses 26 % of FPGA's resource, which lets plenty of space available to implement further preprocessing.



(a) IP counters



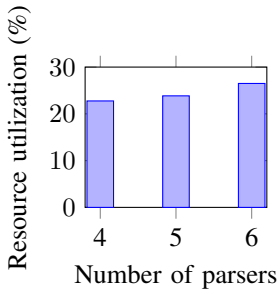
(b) TCP counters



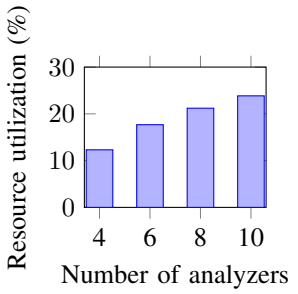
(c) HTTP counters

Fig. 11. Traffic distribution seen by the probe during the dynamic attack detection

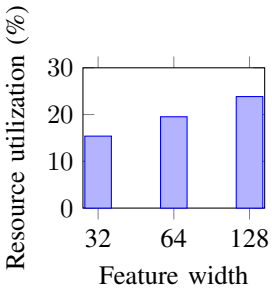
The design latency is an other key component. Each header parser has a latency of 8 cycles and each analyzer has a latency of 6 cycles. Parsers are designed as serial pipelines, so the global latency of header parsing is the addition of each latency. On the opposite, analyzers are parallel, so the global latency of feature analysis is 6 cycles. In the case of the widest design, the latency is 54 clock cycles, meaning 345.6 ns given the frequency. The test solution is built on top of the proposed flexible parser. It is an adaptative probe



(a) Header parser number variation



(b) Feature analyzer number variation



(c) Feature width variation

Fig. 12. Packet parser resource utilization on XC7VX690T

TABLE I
PACKET PARSER SOLUTIONS ON XC7VXH870T

	proposed	[20]	[17]
Datapath width	512	1024	2048
Raw T'put (Gb/s)	160	325	333
Clock period (ns)	3.2	3.154	n.c.
Latency (ns)	96	309	25.9
Slices (% FPGA)	10	12.4	3.9

always tuned for current incoming traffic. In addition to be easily done in Python via an API, parametrization of the packet parser allows the probe to consider the same protocols as in software. This test probe even processes data rates way beyond those processed by its full software counterpart. The presented packet parser combines high performance and high agility.

D. Packet parser architecture comparisons

Table I shows the comparison between the proposed solution with a 512-bit wide datapath and other packet parser designs on Virtex 7 XC7VXH870T. Results are given for a *TCPandIP4andIP6* specialized parser after synthesis for the proposed design and [17], and after implementation for [20]. In order to provide a fair comparison between the related works and the proposed architecture, the parser was configured with 3 header parsers, 3 feature analyzers and 64-bit feature width. This allows parsing of the same protocols, with all the

important features with spare ones. The different solutions are compared in terms of throughput, latency, and resource usage. Flexibility is also discussed in terms of expected agility. Even if this is not measurable, by studying the architecture and its integration, a good idea of its adaptability can be obtained. Despite its high flexibility, the proposed approach has a comparable resource usage with [20], but greatly improves the latency and the flexibility. Comparison with [17] is difficult on a fair basis, since this solution is very dedicated, with almost no flexibility. Resource usage is 2.5 times as large for the proposed solution, but it stays acceptable. Latency is 4 times as large but, once again, remains acceptable at 100 ns, which means a buffering of only 4 kbits. This comparison is done on a simple example and more complex and diverse traffic could lead to compensate the overhead on resource usage by resource sharing between different protocols.

This is compensated by the clear advantage of flexibility, and by the independence from proprietary tools provided by the approach. Adding a new protocol can be done in software only, using the API, while the compared solutions require a new synthesis and configuration of the FPGA. This requires using proprietary tools and stopping current processings on the probe. [20] achieves some level of flexibility, by allowing upgrades with partial reconfiguration by SDNet solution [22]. However, partial reconfiguration requires to reserve space on FPGA for the wider design and imposes strong routing constraints, impacting final performances. It also takes longer than a simple change in parameters.

Another interesting difference lies in the datapath. The compared solutions use a wider datapath. The width of the datapath is linked to the parallelism level of the architecture. This higher it is, the higher the expected throughput is. However, it also creates a constraint on the minimal packet size. If 64 bytes packets at line rate must be processed, the datapath cannot be wider than 64 bytes (512 bits). Saturating the link with 64 bytes packets on the compared solution will lead to packet loss. Higher parallelism also means more duplication in the resources, which might limit the preprocessing.

V. Conclusions and future work

This paper has proposed a packet parser for a novel probe paradigm with a static flexible FPGA design on the datapath and a software part on the control path. This solution achieves a higher agility than conventional smart NICs solutions considering only the reconfiguration of the FPGA. The use of configuration parameters allows to dynamically set protocols and information to extract for further processings while not reconfiguring the probe. This implies that the probe is continuously running and FPGA vendor agnostic. This solution achieves both high flexibility and high data rate, as never encountered before to the authors' knowledge.

The solution has been implemented and deployed on NetFPGA SUME board, based on a Virtex 7 FPGA and only 4 10G interfaces. Future work will focus on the portability of this design on 100 Gb/s links to further increase attainable throughputs. In the current solution, network related algorithms are full-software, and available metadata are limited to basic metrics. Hardware filtering and further offload processings are under way to complete the parser in order to test more elaborated forensic algorithms.

References

[1] OVH, "OVH Mirai attack," accessed: 2017-02-27. [Online]. Available: https://www.ovh.com/fr/a2367.goutte-ddos-n-a-pas-fait-deborder-le-vac

[2] Dyn, "Dyn mirai attack," accessed: 2017-02-27. [Online]. Available: http://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/

[3] E. Bertino and N. Islam, "Botnets and internet of things security," *Computer*, vol. 50, no. 2, pp. 76–79, Feb 2017.

[4] N. Zilberman, P. M. Watts, C. Rotsos, and A. W. Moore, "Reconfigurable network systems and software-defined networking," *Proceedings of the IEEE*, vol. 103, no. 7, pp. 1102–1124, July 2015.

[5] V. Moreno, J. Ramos, P. M. S. del Río, J. L. García-Dorado, F. J. Gomez-Arribas, and J. Aracil, "Commodity packet capture engines: Tutorial, cookbook and applicability," *IEEE Communications Surveys Tutorials*, vol. 17, no. 3, pp. 1364–1390, thirdquarter 2015.

[6] Xilinx, "Xilinx smart networks," accessed: 2017-07-05. [Online]. Available: https://www.xilinx.com/applications/smarter-networks.html

[7] D. Firestone, "Smartnic: Accelerating azures network with fpgas on ocs servers," in *OCP U.S. SUMMIT 2016*, San Jose, CA, March 9-10 2016. [Online]. Available: http://files.opencompute.org/oc/public.php?service=files&t=5803e581b55e90e51669410559b91169&download&path=//SmartNIC%20OCP%202016.pdf

[8] G. Gibb, G. Varghese, M. Horowitz, and N. McKeown, "Design principles for packet parsers," in *Architectures for Networking and Communications Systems*, Oct 2013, pp. 13–24.

[9] S. Gallenmüller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "Comparison of frameworks for high-performance packet io," in *Architectures for Networking and Communications Systems (ANCS), 2015 ACM/IEEE Symposium on*, May 2015, pp. 29–38.

[10] Xilinx, "Xilinx virtex ultrascale+ family," accessed: 2017-02-21. [Online]. Available: https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus.html

[11] Intel, "Intel fpga stratix 10 family," accessed: 2017-02-21. [Online]. Available: https://www.altera.com/products/fpga/stratix-series/stratix-10/overview.html

[12] T. Groléat, M. Arzel, and S. Vaton, "Stretching the edges of svm traffic classification with fpga acceleration," *IEEE Transactions on Network and Service Management*, vol. 11, no. 3, pp. 278–291, Sept 2014.

[13] G. Antichi, C. Callegari, and S. Giordano, "An open hardware implementation of cusum based network anomaly detection," in *Global Communications Conference (GLOBECOM), 2012 IEEE*, Dec 2012, pp. 2760–2765.

[14] J. J. Garnica, S. Lopez-Buedo, V. Lopez, J. Aracil, and J. M. G. Hidalgo, "A fpga-based scalable architecture for url legal filtering in 100gbe networks," in *Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on*, December 2012, pp. 1–6.

[15] A. Bitar, M. S. Abdelfattah, and V. Betz, "Bringing programmability to the data plane: Packet processing with a noc-enhanced fpga," in *2015 International Conference on Field Programmable Technology (FPT)*, Dec 2015, pp. 24–31.

[16] S. Hager, D. Bendyk, and B. Scheuermann, "Partial reconfiguration and specialized circuitry for flexible fpga-based packet processing," in *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Dec 2015, pp. 1–6.

[17] V. Pus, L. Kekely, and J. Korenek, "Design methodology of configurable high performance packet parser for fpga," in *Design and Diagnostics of Electronic Circuits & Systems, 17th International Symposium on*, April 2014, pp. 189–194.

[18] P. Bencek, V. Pu, and H. Kubtov, "P4-to-vhdl: Automatic generation of 100 gbps packet parsers," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2016, pp. 148–155.

[19] G. Brebner, "Packets everywhere: The great opportunity for field programmable technology," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, December 2009, pp. 1–10.

[20] M. Attig and G. Brebner, "400 gb/s programmable packet parsing on a single fpga," in *Architectures for Networking and Communications Systems (ANCS), 2011 Seventh ACM/IEEE Symposium on*, october 2011, pp. 12–23.

[21] G. Brebner and W. Jiang, "High-speed packet processing using reconfigurable computing," *IEEE Micro*, vol. 34, no. 1, pp. 8–18, Jan 2014.

[22] Xilinx, "SDNet," accessed: 2017-02-23. [Online]. Available: https://www.xilinx.com/products/design-tools/software-zone/sdnet.html

[23] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "Netfpga sume: Toward 100 gbps as research commodity," *Micro, IEEE*, vol. 34, no. 5, pp. 32–41, October 2014.

[24] O. Salem, S. Vaton, and A. Gravey, "A scalable, efficient and informative approach for anomaly-based intrusion detection systems: theory and practice," *International Journal of Network Management*, vol. 20, no. 5, pp. 271–293, 2010. [Online]. Available: http://dx.doi.org/10.1002/nem.748

[25] OVH, "Classical attack types on network link," accessed: 2017-02-20. [Online]. Available: https://www.ovh.com/fr/anti-ddos/principe-anti-ddos.xml

[26] [Online]. Available: https://redmine.telecom-bretagne.eu/projects/cyberthd_packetparser