Hardware Implementation of IP Packet Filtering in FPGA

Ana Cholakoska, Danijela Efnusheva and Marija Kalendar
Computer Science and Engineering Department, Faculty of Electrical Engineering and Information Technologies, Ss. Cyril and Methodius University, Karpos II bb, PO Box 574, 1000 Skopje, Macedonia
{acholak, marijaka, danijela}@feit.ukim.edu.mk

Keywords: FPGA, IP Header Fields Extracting, IP Packet Filtering, Network IDS Systems.

Abstract: In the present rapid expansion of the number of computers and devices connected to the Internet, one of the top three issues that need to be addressed is the network security. The greater the number of connected users and devices, the attempts to invade privacy and data of connected users becomes more and more tempting to hostile users. Thus, network intrusion detection systems become more and more necessary and present in any network enabling Internet connections. This paper addresses the network security issues by implementing NIDS style hardware implementation for filtering network packets intended for faster packet processing and filtering. The hardware is based on several NIDS rules that can be programmed in the system's memory, thus enabling modularity and flexibility. The designed hardware modules are described in VHDL and implemented in a Virtex7 VC709 FPGA board. The results are discussed and analyzed in the paper and are presenting good foundation for further improvement.

1 INTRODUCTION

Many concepts of security measures for computer communication networks have been developed over the years. Consequently, it has been shown that some of them are more effective when it comes to the resilience to various network intrusions and attacks in comparison to other known systems. Network Intrusion Detection Systems (NIDS) allow greater control over the traffic generated in the network while applying several mechanisms and rules for filtering known and sometimes predicting unknown types of network attacks according to anomalies detected in the monitored network traffic.

Naturally, these kind of IDS systems are generally software defined, and are still vulnerable to unknown or novel types of attacks. Nevertheless, the software defined NIDS systems are quite flexible, modular and easily upgradeable. Despite the flexibility, the main potential liability of these software based NIDS systems is their inability to handle and process the continuous and daily increasing quantities of network traffic.

Consequently, the concept of filtering network packets in this paper has been based on an existing software system for protection against unauthorized intrusions. Namely, SNORT – a network IDS system, is well known for its ever evolving architecture and the vast collection of rules for detecting unwanted network traffic. Precisely those rules are taken as the basis for the hardware implementation and the packet filtering tests.

Despite the software solutions, several specialized hardware solutions intended for packet filtering have also been proposed, in order to bring additional speed to the process of filtering. Regarding hardware network packet processing, one of the most popular and vastly used solutions are the Network processors (NPs) [1]. In general, they represent devices specially tailored to perform various network processing operations: header parsing, bit-field manipulation, pattern matching, table look-ups, and data movement, [2]. Similarly, one of the more renowned and studied architectures of network packet processing is the NetFPGA architecture [4]. NPs are usually used in different types of network equipment, including routers, switches, IDS or firewalls, [3]. Accordingly, NPs spend significant part of processor cycles on packet header parsing, especially when the packet header fields are non byte- or word-aligned. Improving the number of processor cycles needed for packet header parsing has been addressed in our previous work [8], enabling a single-cycle memory access to these non byte- or word- aligned header fields. The simulation results and the flexibility of the proposed solution

were investigated utilizing a reconfigurable hardware platform Virtex7 VC709 FPGA.

Resuming this previous work and building on, after the network packet headers have been parsed and accessed in memory, this paper investigates the possibilities for implementing software IDS packet filters in hardware. Consequently, the primarily goal of this paper is to augment software defined IDS systems by implementing and simulating network packet filtering modules in hardware, using the faster hardware resources of an FPGA board while retaining the flexibility of the software based rules. Such hardware/software co-design would bring speed, as well as flexibility while implementing and applying the rules for network packets filtering.

The rest of this paper is organized as follows: Section II gives an overview of the state of the art in the area presenting different network processing and filtering hardware solutions. Section III describes the design of the rule filtering hardware intended for increased security and layouts the benefits of the hardware design and the flexibility due to the programmability of the FPGA system. Section IV presents the additional hardware module for extracting/ writing ip header fields from/to memory in a single-cycle access, especially for non byte- or word- aligned packet header fields. Section IV presents simulations and synthesis results from the FPGA implementation of the IP packet filtering hardware module in VHDL. Section V concludes the paper, outlining the benefits of the proposed IP packet filtering module.

2 STATE OF THE ART

Contemporary technology advances increase the pace of rapid expansion of the number of computers and devices connected to the Internet on daily basis. As a result, one of the highest priority issues that need to be considered in this enormous network is the network security. The greater the number of connected users and devices, the attempts to invade privacy and data of connected users becomes more and more tempting to hostile users. Thus, Network Intrusion Detection Systems (NIDS) become more and more necessary in any network connected to the Internet, and are taking the lead in the battle against intruders.

In order to enhance the security NIDS have to inspect incoming network packets looking for unwanted and hostile traffic. This is foremost done via various software platforms (e.g. Snort), but the major increase in daily network traffic imposes a great challenge for software platforms. Therefore, many researchers have already turned to hardware designs for many network issues, including security.

One of the first topics for hardware designed processing is aimed to network packets header parsing, which is a prerequisite for the packet filtering operations.

Many researchers have been working in both areas since they are interconnected. Namely, the process of identifying and extracting fields from a packet header and doing it in hardware for faster processing is being addressed in many works [7], [9]. In most cases NPs are used to perform fast packet processing where the IP header is being processed, by analyzing, parsing and modifying its content, [3]. NPs might include some specialized hardware units to perform classification of packets, lookup and pattern matching, queue management and traffic control which on the other side can be used for the purpose of packet filtering for security reasons. In recent research, NP software is getting closer to the NP hardware, such as in [10] where part of the packet processing tasks such as classification or security are offloaded to application-specific coprocessors. Other proposals make big use of FPGA technology for packet parsing, enabling implementation of pipeline architectures and thus achieving high-speed network stream processing, [12]. Actually, the reconfigurable FPGA boards can be used to design flexible multiprocessing systems that adjust themselves to the current packet traffic protocols, which in turn makes them very suitable for packet filtering regarding security.

FPGA technology is widely used in combination with NIDS and packet filtering for security purposes. For example, the authors in [15] propose a modular approach for grouping homogeneous traffic, and then splitting it for filtering in different specialized hardware blocks, each supporting a (smaller) rule set tailored for the specific traffic category. The rule sets are based on the well known Snort NIDS. The paper concludes that the exploitation of traffic classification and load statistics may bring significant savings in the design of HW NIDS. Similarly, [18] introduces a packet pre-filtering approach, based on the observation that very rarely a single incoming packet fully or partially matches more than a few tens of IDS rules. Finally, packet pre-filtering prevents matching at least 99% of the SNORT rules per packet, thus minimizing processing time and improving the scalability of the system.

Other reconfigurable FPGA approaches include [16] and [19] who propose modular and programmable hardware accepting configuration changes (for the rules) in real time, while processing the important packet header fields and/or payload.

Finally, [20] investigate an approach suitable for current and future high speed networks of 100 Gb/s and more, by trimming the traffic that needs to be filtered by using FPGA-based packet filters. The goal is achieved by implementing a new "network grammar" for specifying protocols and filtering rules for continuous stream of data. The grammar compiles directly to Verilog code for packet filters. The new concept was tested on two proof-of-concept designs: a DNS filter and a simple firewall.

3 RULE FILTERING HARDWARE FOR INCREASED SECURITY

Considering network IP packets processing, usually the main operation is packet header parsing, mostly for packet routing purposes. Nevertheless, since the header fields of the packet have been already extracted, further processing is possible and very useful for the aim of traffic engineering, traffic shaping, network security.

This paper is taking into account further packet processing regarding security, i.e. applying specific IDS software rules on received network packets for the purpose of filtering unwanted packets from the network. The IP header that is received at the input of the previous module [8], is usually immediately written in memory without previously going through any inspection. Consequently, to be able to enforce greater security, in this architecture, a block for filtering IP headers is being implemented previous to writing the packet into memory. The filtering block uses rules previously defined by the system administrator (in the considering case, predefined rules from the Snort IDS for checking IP headers). The corresponding rules have been programmed into memory on a particular location.

Figure 1 presents a proposed scheme for an implementation of such IP packets filter that enables applying appropriate rules, therefore filtering the network packets. Following the packet parsing module, a selected IP header field is being input in this proposed module (whether a TTL, protocol etc.) and is subsequently compared to the rule in memory. If these two fields match, then the Alert signal outputs a high level signal, which in turn allows the appropriate field to be written in memory. The appropriate location is calculated through the BaseAddress and MemoryOffset information. On the contrary, if these fields do not match, the package is rejected and not written in memory. This is signalled again through the Alert signal that outputs a low level signal.

Initially, the Alert signal may not be present, however this kind of signalling proves very useful to the previous module keeping the headers (or the entire) IP package. The value of the Alert line would signal the IP packet parser module that in fact there is at least one field of the header that has not passed the filter, enabling it to make a rejection of the entire packet header and delete all packet fields.

Figure 2 presents a simulation conducted with the HDL programming package VIVADO showing the signal layout for an unsuccessful rule matching procedure. If the incoming IP packet header field does not match the expected value in the filter field, the Alert signal outputs a low level signal value (0 value), indicating the incoming packet field is invalid.

Figure 3 presents the contrary situation where the incoming IP packet header field matches the rule, and the Alert signal outputs a high level signal value (1 value) , thus indicating that the IP packet header field passes the security check and can be saved in memory.

The presented hardware solution enables faster hardware comparison of the IP packet headers against the programmed rules compared to the software approach. Moreover, programming the filtering rules in memory imposes great flexibility and possibility for implementing and testing new future rules for new and emerging threats. The presented hardware module is a proof of concept solution requiring further investigation in the area of hardware rule optimization, as well as field comparison optimization.

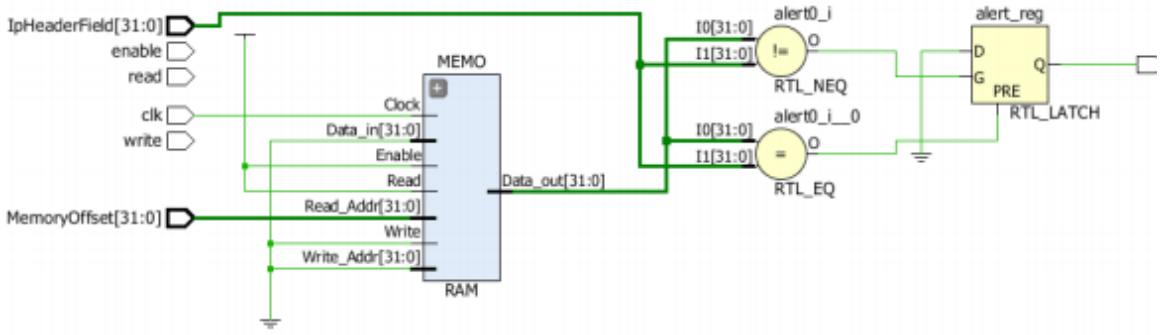

Figure 1: Scheme of the implementation of an IP filter.

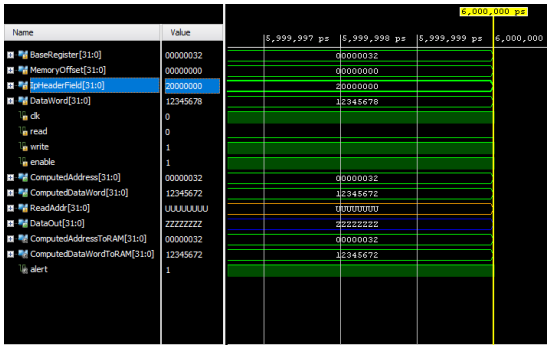

Figure 2: Field filtering - non matching.

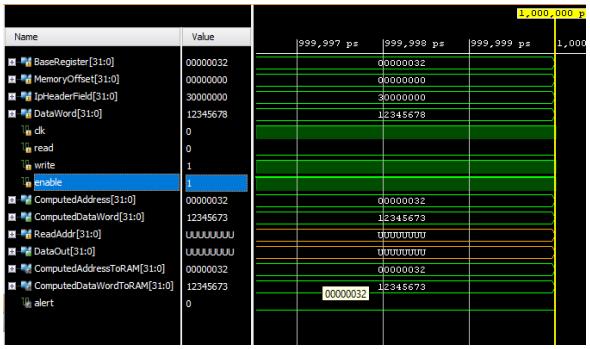

Figure 3: Field filtering – matching.

4 HARDWARE FOR WRITING IP HEADER FIELDS IN MEMORY

Besides the concept of IP header fields filtering, this paper details and proposes specialized hardware modules for IP packet header fields extraction before processing, and subsequently IP packet header fields rewriting after processing. These modules help the direct extraction of the needed IP packet field in a single memory access operation, further improving the speed for information transfer from/to the processor as proposed in our previous work [8]. The IP packet fields extracted to the processor are then used in all other processing (filtering) operations. In order to return the corresponding field to the proper memory location (proper location in the), several reversed operations have to be performed. This section details these operations.

After the processing/filtering of the specified IP header field, the field should be returned to the correct position in the complete IP packet header in memory. The signals lines augmenting this process are BaseRegister, MemoryOffset and DataWord, as well as the IpHeaderField containing the shifted address to the processed IP header field. The BaseRegister carries the initial IP packet address into the memory, MemoryOffset tells which header field is being considered (order number). Than this value goes through a Look-up table where the order numbers of the fields are paired with one of the six words on the from the IP header. Every header field is mapped with the word that it is contained in. For example the field “Time to Live” is field 7 (it starts with 0) and it is mapped with the word 2 represented in 32 bits (it starts again with a count of 0). The values from BaseRegister and MemoryOffset are summarized and thus the address where the new value should be written is calculated.

The order number of the word containing the IP header field under consideration, arrives to the decoder, which in turn decides which of the FieldLogic fields to activate. IpHeaderField and DataWord are also present enabling the assembly of the IP header word with the selected order number. At this point, having the needed data and the address in memory, the content of the IP header word containing the processed IP header field is written in memory.

To illustrate the process a selected example will be explained taking into account the “Time to Live” field. “Time to Live” field is the seventh header field in order and it is contained in the second word (counting from 0). Let the value of this second word

be DataWord = 12345678 (hexadecimal). The hardware would extract only the TTL field and shift its value to the left, resulting in IpHeaderField = 78000000. "BaseRegister" represents the first address of the IP packet header in memory, and for ease of the example calculation it can be freely chosen. So let's assume BaseRegister = 00000000. The initial value of MemoryOffset = 00000007. Then, by searching through the Look-up table, the value 00000002 will be output and stored in the MemoryOffset field (the seventh field is in the second word of the header). Now let's assume that the FieldLogic module initiates some change in the IpHeaderField resulting in a new field value IpHeaderField=77000000. The correct memory address where this new value has to be rewritten is calculated from the existing values:

BaseRegister + MemoryOffset =
= 00000000 + 00000002 = 00000002

thus marking that the field should be rewritten in the second word of the header. Finally, the correct data from the DataWord and the new IpHeaderField need to be combined and written back to the correct place.

The FieldLogic module is responsible for correctly combining the DataWord and the IpHeaderField using the order number of the IP header field (7 in the example). Consequently, the seventh part of the FieldLogic module would be activated, enabling shifting and adding logic to finally calculate the new value of the DataWord (incorporating the IpHeaderField at the last two "digits"). Finally the new and correct DataWord exiting the FieldLogic module can be written at the correct address calculated previously.

Since the finest details regarding the IP header fields extraction and rewriting are "hidden" in the FieldLogic module, Figure 4 and Figure 5 give details regarding the hardware realization of the FieldLogic parts designed for the Version IP header field and the Header Checksum IP header field, just for the purpose of illustration. As it can be seen from these figures, the number and position of the bits in the IP packet header influences the complexity of the hardware intended for IP header field extraction/writing. Version IP header field is 4 bits long, and positioned at the end of word 1. Hence, several operations of shifting, multiplexing and additions are needed for the operations. On the other hand, the Header Checksum IP header field is 16 bits long and is byte aligned in word 3 of the IP packet header. Consequently, the needed hardware is less complex. This is true for all the other IP packet fields with similar observations.

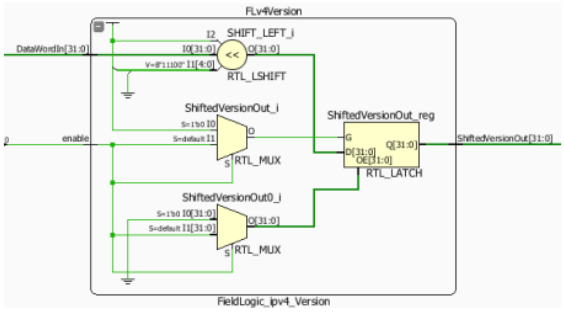

Figure 4: Field_logic_ipv4 Version design.

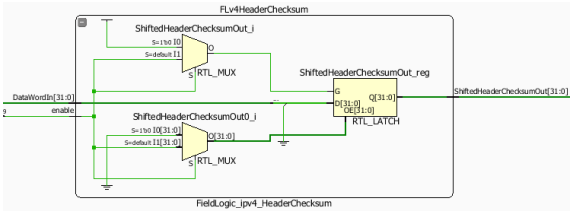

Figure 5: Field_logic_ipv4_HeaderChecksum design.

The last field of the IPv4 header is the SourceAddress field (the sender of the packet). In this case, the field has a size of a one word, or 32 bits. Therefore, since this field can be transferred completely to memory, and on the right position, no additional selection is performed for certain bits.

As presented, specialized hardware is needed for each IP packet header field, and special FieldLogic parts have been designed for all IP packet header fields for the IPv4 and IPv6 packet headers.

Regarding IPv6 packet header, the fields: Version, Traffic Class, Flow Label, Payload Length, Next Header, Hop Limit are mapped similarly as the IPv4 fields. Interesting cases with IPv6 are the Source and Destination Address fields taking 128 bits each.

5 USE OF RESOURCES

completing the functional simulation of the filter and parser of the IP headers, comes the FPGA synthesis and the implementation of the device itself.

The results of the synthesis presented in Figure 6 indicate that the proposed filter for IP network packets can be implemented on in Virtex7 VC709 evaluation platform by utilizing less than 0.01% of the slice registers and 0.16% of the slice LUT resources, which represents less than 1% of the possible FPGA resources. As it is obvious from the

low FPGA resource's utilization, the initial IP header filtering logic design can be further extended and then implemented in the same Virtex 7 VC 709 FPGA board.

This means that as further research, the proposed IP packet filter can be expanded by incorporating more packet header fields and/or packet payloads, as well as additional and more complex packet header filters, all implemented on the same FPGA platform. From the ability to reconfigure the FPGA device, it can be concluded that this kind of module can be very easily adapted to work with other protocols, which in turn indicates great flexibility and low cost.

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	689	0	433200	0.16
LUT as Logic	689	0	433200	0.16
LUT as Memory	0	0	174200	0.00
Slice Registers	36	0	866400	<0.01
Register as Flip Flop	0	0	866400	0.00
Register as Latch	36	0	866400	<0.01
F7 Muxes	0	0	216600	0.00
F8 Muxes	0	0	108300	0.00

Figure 6: Used resources of Virtex 7.

6 CONCLUSIONS

This paper addressed a very current and vastly spread issue of network security that affects more and more large institutions and companies every day. Nowadays, this topic of research is placed on the top three priority places and great efforts are being made for it to be improved. Building from here, the idea for a combined software/hardware solution for better network protection from unauthorized intrusions.

Firstly, the concept of filtering network packets has been based on some of the existing software systems for protection against unauthorized intrusions. As the basic software solution, one of the widely used open source systems has been chosen. Namely, SNORT – a network IDS system, is well known for its ever evolving architecture and the vast collection of rules for detecting unwanted network traffic. Exactly those rules are input as the basis for hardware implementation.

In order to be able to offer a complete hardware and software solution, building on top of Snort, a VHDL hardware design was implemented and tested. The hardware design encompasses a packet filter based on hardware implementation of Snort rules, as well as a hardware accelerator for IP packet header fields extraction and rewriting.

The hardware design was implemented on the Virtex 7 VC709 FPGA board, thus proving the functionality and envisioning the future possibilities for improvement. The realization of the module for network packet filtering, presented solid results showing that the proof of concept filter can be implemented by using only <0.01% of the slice registers and as little as 0.16% from slice LUT resources, which represents less than 1% of the possible FPGA resources in total.

The results obtained in this manner indicate the great flexibility and low cost of the module, as well as the possibility for its expansion towards filtering different types of packages, protocols, packet behaviour, as well as adding additional selection filters.

As for the future development of this module, one of the possibilities is to design an additional module for separating the header from the packet, and enabling parallel processing of both, regarding network processing and filtering for increased security reasons.

REFERENCES

[1] B. Wheeler, "A new era of network processing," LinleyGroup Bob Wheeler's White paper, 2013.

[2] P.C. Lekkas, "Network Processors: Architectures, Protocols and Platforms," McGraw-Hill Professional, 2013.

[3] R. Giladi, "Network Processors - Architecture, Programming and Implementation", Ben-Gurion University of the Negev and EZchip Technologies Ltd., 2008.

[4] J. Naous, G. Gibb, S. Bolouki, N. McKeown, "NetFPGA: reusable router architecture for experimental research", in Sigcomm Presto Workshop, 2008.

[5] B. Doud, "Accelerating the data plane with the Tile-mx manycore processor", in Linley Data Center Conference, 2015.

[6] J. M. P. Cardoso, M. Hubner, "Reconfigurable Computing: From FPGAs to Hardware/Software Codesign", Springer-Verlag, 2011.

[7] G. Gibb, G. Varghese, M. Horowitz, N. McKeown, "Design principles for packet parsers", In ACM/IEEE Symposium on Architectures for Networking and Communications Systems, 2013, pp. 13–24.

[8] D. Efnusheva, A. Tentov, A. Cholakoska, M. Kalendar, "FPGA Implementation of IP Packet Header Parsing Hardware", In Proc. of the 5th International Conference on Applied Innovations in IT, (ICAIIT), 2017, pp. 33-41.

[9] J. Kořenek, "Hardware acceleration in computer networks". In 16th International Symposium on

Design and Diagnostics of Electronic Circuits Systems, 2013.

[10] L. Kekely, V. Puš, J. Kořenek, "Software Defined Monitoring of application protocols", In IEEE Conference on Computer Communications, 2014, pp. 1725–1733.

[11] R. Bolla, R. Bruschi, C. Lombardo, F. Podda, "OpenFlow in the Small: A Flexible and Efficient Network Acceleration Framework for Multi-Core System", In IEEE Transactions on Network and Service Management, 2014, pp. 390-404.

[12] V. Puš, L. Kekely, J. Kořenek, "Design methodology of configurable high performance packet parser for FPGA", In 17th International Symposium on Design and Diagnostics of Electronic Circuits Systems, 2014, pp. 189-194.

[13] M. Attig, G. Brebner, "400 Gb/s Programmable Packet Parsing on a Single FPGA", In Seventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems, 2011, pp. 12-23.

[14] G. Brebner, W. Jiang, "High-Speed Packet Processing using Reconfigurable Computing", In IEEE Micro, vol. 34, no. 1, 2014, pp. 8-18.

[15] S. Pontarelli, G. Bianchi, S. Teofil, "Traffic-aware Design of a High Speed FPGA Network Intrusion Detection System". In IEEE Transactions on Computers, Vol. 62, Issue: 11, 2013, pp. 2322 - 2334.

[16] R. Ajami, A. Dinh, "Embedded Network Firewall on FPGA", In Proc. of 8th 2011 International Conference on Information Technology: New Generations, 2011.

[17] S. Yusuf, W. Luk, M.K.N. Szeto, W. Osborne, "UNITE: Uniform hardware-based Network Intrusion deTection EngineS". In Proc. of ARC 2006: Reconfigurable Computing: Architectures and Applications, 2006, pp 389-400.

[18] I. Sourdis, V. Dimopoulos, D. Pnevmatikatos, S. Vassiliadis, "Packet Pre-filtering for Network Intrusion Detection". In Proc. of ANCS'06, 2006.

[19] A. Wicaksana, A. Sasongko, "Fast and Reconfigurable Packet Classification Engine in FPGA-Based Firewall", In Proc. of 2011 International Conference on Electrical Engineering and Informatics, 2011.

[20] J.F. Zazo, S. Lopez-Buedo, G. Sutter, J. Aracil, "Automated synthesis of FPGA-based packet filters for 100 Gbps network monitoring applications", In Proc. of 2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig), 2016.