Improving a Network Security System by Reconfigurable Hardware

Shaomeng Li, Jim Torresen, Oddvar Søråsen
Department of Informatics, University of Oslo, N-0316 Oslo, Norway
shaomenl, jimtoer, oddvar@ifi.uio.no

Abstract

Improving network security systems by using reconfigurable hardware is an important research field since the speed of the Internet is increasing. In this work, we have implemented stateful TCP inspection in a Field Programmable Gate Array (FPGA) to help alleviating a bottleneck in network intrusion detection systems (NIDSs). Today's software based NIDSs (e.g. Snort) show inefficiency and even fail to perform for the faster Internet. Implementing stateful TCP inspection in FPGA aims at achieving an efficient, fast NIDS in general, — Snort specifically. By dividing a TCP connection into two flows, one to the Server and another to the Client, monitoring of the TCP flow can be sped up. A TCP flow deals not only with a single packet but also with multiple packets over the network. Reassembly of those packets is one of the main tasks that the TCP flow monitoring should accomplish. By parallelizing the tasks of reassembling TCP packets on the Server and the Client on an FPGA, the performance of stateful TCP inspection can be greatly improved. The performance obtained by this work is a throughput of 2.75 Gbps.

1 Introduction

A Network Intrusion Detection System (NIDS) is a more advanced security tool than a firewall in a network security system [1]. "Stateful inspection" is applied in NIDSs. It is used for checking the handshakes in a communication session by using detailed knowledge of the rules of the communication protocol. This is to make sure that it is completed in an expected and timely fashion. By checking a connection (packet by packet) – not just one single packet, and knowing what has just happened and what should happen next, stateful inspection detects incorrect or suspicious activity and alerts flags to the system administrator.

This work examines how this challenging and time critical task can be off-loaded from software to reconfigurable hardware, field programmable gate array (FPGA). A short description of stateful TCP inspection is given in section 2. The paper continues to describe the hardware architecture in section 3. It is based on our previous work [2] and shows in section 4 the experiments recently done on the FPX platform at Washington University in Saint Louis to verify the operation. Results show that the circuit functions correctly and that a speed of 2.75 Gpbs can be obtained. The paper ends by a summary of our achievements.

2 Stateful TCP inspection

TCP (Transmission Control Protocol) [3] is an important internet protocol. It provides a full duplex reliable stream connection between two end points in the TCP/IP network. The approach of using stateful inspection will be one of the best ways (maybe the only way) to monitor a TCP connection.

The main function of stateful inspection in a TCP connection is firstly to track the states of the connection within one session to be able to synchronize the handshake signals. Two end systems are associated with the connection process: the Server side and the Client side. A TCP three way handshake is used so the two end systems can establish a connection between themselves. When the three way handshake is properly established, the Server and the Client can exchange data. From there on, each end system processes the data from the other end system. That is, the Server processes the data from the Client and the Client processes the data from the Server, respectively. Thus, the Server and the Client TCP stream reassembly in an NIDS can be performed independently. Apart from the handshakes, this is the second function needed to be performed in the module of stateful TCP inspection.

Sliding window algorithms can independently be used in the Server and the Client TCP stream reassembly units for efficiently processing packets. Data path processing in parallel is the main feature used when implementing the Server and the Client TCP stream reassembly in FPGA. Thereby, the performance of NIDS could be enhanced. Processing the data flow on the Server side and the Client side in parallel and fully considering context information of the TCP connection are our main contributions to improve processing of TCP connections in a NIDS.

2.1 Comparison with related works by analyzing TCP context information

Comparison with related works [4] [5] was made in the view of TCP context information. Context information for a TCP flow mainly consists of the IP address, TCP port number, sequence number, acknowledgment number, window size, TTL and TCP flags, both in the Server and in the Client [3]. We denote this context information as IPs, IPc, Ports, Portc, Seqs, Seqc, Acks, Ackc, Wins, Winc, TTLs, TTLc, TCPflags, and TCPflagc, where s refers to the Server and c to the Client.

Each piece of context information has its essential function facilitating a network TCP monitor. Omitting some of the context information in an FPGA TCP network monitor consequently affects the functionality of the monitor. We would therefore argue that a network TCP monitor should preferably take into account all the TCP context information. The ability to monitor all context information should be a minimum benchmark of any TCP network security systems in reconfigurable hardware.

We can break the analysis of TCP context information down to the following: IPs, IPc, Ports and Portc are used to define a TCP connection. These four pieces of the context information are however not sufficient for the job of analyzing a net-

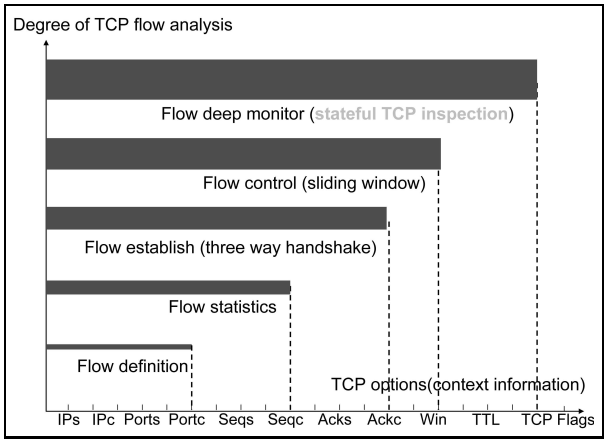

Figure 1. TCP context information vs. degree of TCP flow analysis.

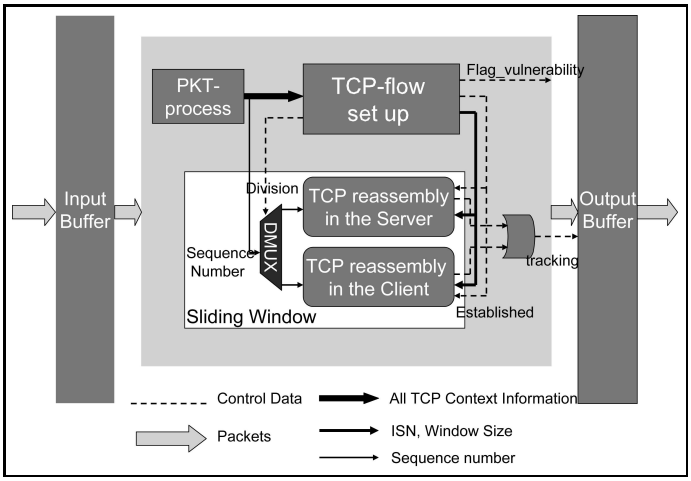

Figure 2. Stateful TCP inspection on FPGA.

work TCP connection. The sequence number is added for simple TCP flow statistics, however it is not enough information to establish a TCP flow using the three way handshake algorithm. The acknowledge number and TCP flags (the ACK and the SYN) are further needed to set up a connection properly. After the connection is established, flow control can be done by tracking the window size and applying the sliding window algorithm. The TTL and TCP flags (the RST and the FIN) are used to tear down the connection. The stateful TCP inspection proposed in this work is based on all the listed context information parameters to gain a fully functional network security system. Using the features of reconfigurable hardware and the proposed architecture, the functionality and performance of a TCP network monitor is enhanced. However, approaches of other related works [4] [5] are limited because they work on the level designated as "Flow definition" or "Flow statistics" resulting in an incomplete analysis over a network TCP monitor. The illustration in Figure 1. shows the degree of functionality which can be obtained vs. TCP context information.

2.2 STREAM4 in Snort

NIDS Snort [6]– a software based NIDS with 3000 lines of software code, is designed to conduct the TCP stateful inspection, performing two functions: Stateful inspection sessions (monitoring handshakes) and TCP stream reassembly (collecting together packets belonging to one TCP connection). This is called a STREAM4 preprocessor. Testing Snort on various networks has shown that the STREAM4 preprocessor leads to a bottleneck in Snort for some network traffic environments (details can be found in [7]). Thus, to improve the performance of Snort, we would like to explore a new architecture in reconfigurable hardware for implementing STREAM4.

3 An architecture for stateful TCP inspection in reconfigurable hardware

The block diagram of the stateful TCP inspection architecture is given in Figure 2. An incoming packet with 32 bit width is input to the reconfigurable hardware unit which processes the TCP three way handshake and the Server and Client TCP stream reassembly.

The purpose of the input buffer unit is to store packets which have been sent from one end point, but have not yet been acknowledged by the other end. In addition, this unit stores delayed, duplicated, misread or retransmitted packets. In this case, boundary flops were used for simplicity sake.

In the PKT-process (packet process) unit, all context information of TCP packets (IP, Port, Sequence number, Acknowledgment number, Window size, TTL, TCP flags) is tracked and stored in registers for downstream processing in this system.

The TCP-flow connection unit is implemented as a state machine to check the three way handshakes of the TCP connection. Five important states (CLOSED state, SYN-SENT state, SYN-RECV state, ESTABLISHED state and EXCHANGE state) are the main states to be examined to build up the proper TCP three way handshakes needed for the TCP connection. During the building of the TCP connection, the control signals Division, Flag-vulnerability and Established will be the output to the downstream units. The latter two signals are rather self-explanatory. The Division signal is set to '1' while packets are sent from the Client side and to '0' while packets are sent from the Server. The Division output therefore can be used to control the DMUX unit to separate the sequence numbers of the incoming packets. In this process, attacks such as Stealthyscan and half TCP connection can be identified. The 32 bit de-multiplexer (DMUX) is applied to separate the sequence numbers of incoming packets into two categories: one is data flow from the Server (or to the Client), another is data flow from the Client (or to the Server). The reason for doing this is to perform the TCP reassembly in the Server unit and the TCP reassembly in the Client unit, respectively in parallel. The processing of the TCP flow control parts in a NIDS is therefore accelerated, of course, at a cost of extra FPGA resources.

Two 32 bit comparators and one 32 bit adder are needed to implement one TCP reassembly unit, as seen in Figure 3. If the sequence number of an incoming packet is outside the band size (band size is decided by the ISN number and window number), the packet will be dropped. Packets both from the Server and the Client can otherwise be loaded into the output buffer using the tracking signal from the Client and Server reassembly units respectively. A 16 bit window size is obtained by acknowledging the packet from the destination end when the transition reaches at the SYN-RECV state and the ESTABLISHED state in the TCP-flow connection unit. The sliding window algorithm can therefore be accomplished in these two reassembly units.

The size of the output buffer unit is not a critical issue. However, a dual port (write/read) memory is required for the output buffer unit, because a dual port memory is able to receive new

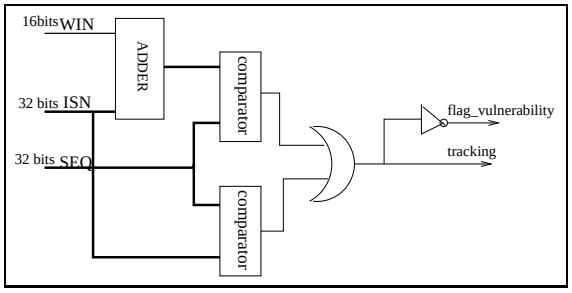

Figure 3. TCP reassembly unit.

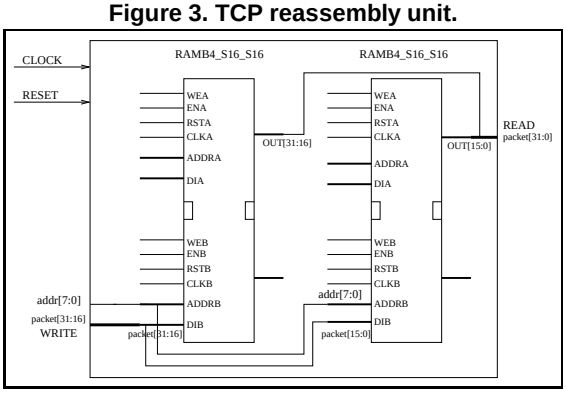

Figure 4. Output buffer unit.

data while sending data. Thus a dual port block RAM is used to implement the output buffer unit.

The Virtex XCV2000E-6 FPGA used in this work contains RAM blocks called SelectRAMs. Each one has a capacity of full synchronous dual port 4096-bit memory and is ideal to implement the output buffer unit. One such block SelectRAM can be configured as a memory with different data widths and depths. However, since a dual port RAM is used, the maximum data width is limited to 16 bits. The library primitive, RAMB4-S16-S16 [8] is dual ported where each port has a width of 16 bits and a depth of 256 bits. It is available for the XCV2000E-6. By considering the size of the RAMB4-S16-S16 and the packet which has 32 bit data width, two such RAMB4-S16-S16 primitives are therefore needed to implement *one* 32 bit data dual port memory. And one RAMB4-S16-S16 needs two SelectRAMs. Figure 4. shows the unit of the output buffer of the stateful TCP inspection architecture.

4 Experiments

We conducted the testing of the designed architecture of stateful TCP inspection on an FPX (Field-programmable Port Extender) platform, an advanced network reconfigurable hardware research platform at Washington University in Saint Louis (WUSTL) [9]. We implemented stateful TCP inspection in a Xilinx XCV2000E FPGA on the FPX platform. Figure 5 shows the set-up for the prototype. WUGS-20 is an ATM switch with 8 ports developed at WUSTL. Internet traffic passes through WUGS-20 which mirrors the data to an FPX where the stateful TCP inspection is implemented as one application of the IP wrapper [9]. The Internet traffic passes to a router as well, which contains a fast Ethernet switch connected to workstations. Data from workstations also pass to the router, then to the Internet through the WUGS-20. In this experiments, NCHARGE was used to send network traffic to the Internet and monitor incoming traffic. NCHARGE stands for Networked Configurable Hardware Administrator for Reconfiguration and Governing via End-systems, provides a stan-

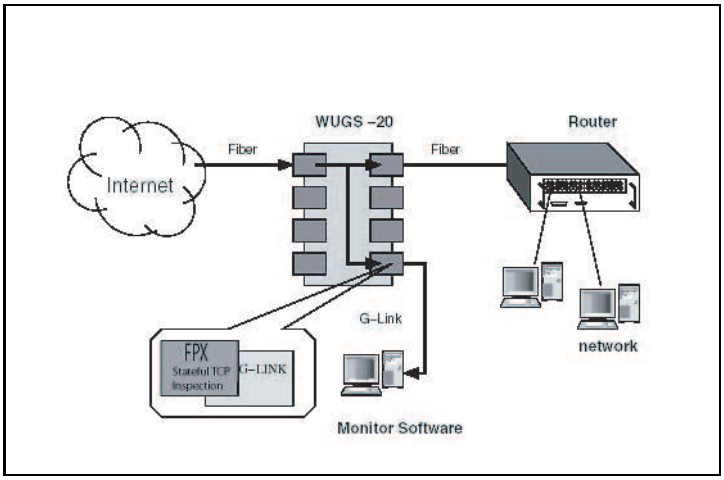

Figure 5. Testing Layout For Stateful TCP Inspection

dard Application Programming Interface (API) between software and reprogrammable hardware modules. Using this API, multiple software processes can communicate to one or more hardware modules using standard TCP/IP sockets. NCHARGE also provides a web-based user interface to simplify the configuration and control of an entire network switch (WUGS-20) that can contain several software and hardware modules (in this case, one hardware module). Because the stateful TCP inspection module monitors TCP packets on the network, only TCP format packets were sent to the FPX for being tested. We performed functional testing by doing experiments to observe the real performance of stateful TCP inspection in terms of flag-vulnerability signal alarming in the monitor PC. If a TCP connection was built properly, incoming packets belong to the actual TCP connection, and the monitor PC was not alarmed by the signal of flag-vulnerability. Otherwise the PC was alarmed.

4.1 Results

Our implementation of this design was analyzed using the ISE FPGA tool from Xilinx. The design was mapped to the FPGA XCV2000E-6 on the FPX research platform for testing. All individual modules such as the PKT-process unit which does packet parsing, the TCP-flow connection unit, TCP reassembly unit, and DMUX unit are implemented in VHDL. The simulation of those functions was conducted under the simulator of Modelsim XE II v6.5a from Xilinx.

The whole system has been placed and routed into the XCV2000E FPGA. The minimum clock period for data from input to output is 11.62 ns which corresponds to a throughput of 2.75 Gbps. A total of 2.5 percent of SLICEs (496 of 19600 SLICEs) in XCV2000E-6 is used to implement this system.

However, in TCP/IP networks, the Server often needs to be able to handle multiple connections simultaneously. Hence, multiple TCP connections have to be considered. We discuss this in the following subsections.

4.1.1 One TCP connection

For doing analysis of the multiple TCP connections, the testing results of one TCP connection which we got from the FPGA place and route report are important to list, as seen in Table 1.

4.1.2 Multi TCP connections

From Table 1, the process which consumes most SLICEs in the FPGA is the module of doing the TCP three-way hand-

Stateful TCP inspection	SLICEs
TCP-flow connection	166
TCP reassembly in Ser	51
TCP reassembly in Clt	51
Output buffer unit	8
PKT-process	145
32 bit DMUX	37

Table 1. Statistics of Modules of one TCP connection.

shake (TCP-flow connection) using 166 SLICEs. Although there are 19600 SLICEs in one XCV2000E-6 FPGA, the possibility of having multiple TCP connections is limited to the capability of implementing the output buffer unit in one such FPGA. The reason for this is that the height of the Configurable Logic Block (CLB) array in one FPGA decides the number of block SelectRAMs that can be implemented. One XCV2000E-6 FPGA with the amount of 160 block SelectRAMs can therefore implement only 40 TCP connections. And the remaining logic resources of the XCV2000E-6 FPGA except block SelectRAMs can implement the remaining modules of 40 TCP connections simultaneously. When one new FPGA is used to implement multi TCP connections, apart from counting the number of block SelectRAMs, the designer should also be sure that the size of SLICEs of the new FPGA is checked to see whether it is enough logic to implement remaining modules of the multi TCP connections simultaneously.

5 Conclusion

Stateful inspection of a TCP connection is studied and implemented on FPGA based hardware to remove the bottleneck of TCP connection in a network traffic environment. A novel approach using reconfigurable hardware is introduced. Experiments show that the system functions correctly and that the performance could be improved by this implementation to a throughput of 2.75 Gbps.

A NIDS, where the user can turn on or off certain processing intensive modules like string matching or stateful TCP inspection depending on the application environment, is expected to be a part of future network security systems. The FPGA-based reconfigurable hardware can offer this type of promising strategy. Our future work will be devoted to exploit such a NIDS.

Acknowledgement

It is a pleasure to thank Dr. John Lockwood and the Applied Research Lab (ARL) at Washington University in Saint Louis for hosting the first author's visit in January-June 2004, during which time this work was carried out. Thanks to all members of ARL for help and support.

Special thanks to the Research Council of Norway and the Department of Informatics, University of Oslo for funding this visit.

References

[1] Shaomeng Li et al. *"Exploiting Reconfigurable Hardware for Network Security"*. in Proc. of 11th Annual IEEE Symposium on Field-Progammable Custom Computing Machines(FCCM'03), 2003.

[2] Shaomeng Li et al. *"Exploiting Stateful Inspection of Network Security in Reconfigurable Hardware*. in Proc. of 13th International Conference on Field Programmable Logic and Application (FPL2003), Lisbon, Portugal, September, 2003.

[3] J. Postel. *"Request For Comment 793, Transmission Control Protocol"*. 1981.

[4] David V. schuehler et al. *"TCP-Splitter: A TCP/IP Flow Monitor in Reconfigurable Hardware"*. in Proc. of Hot Interconnects 10 (HotI-10), Stanford, CA, Applied Research Laboratory, Washington University, August, 2002.

[5] Marc Necker et al. *"TCP-Stream Reassembly and State Tracking in Hardware"*. in Proc. of 10th Annual IEEE Symposium on Field-Progammable Custom Computing Machines(FCCM'02), School of Electrical and computer Engineering, Georgia Institute of Technology, Atlanta, GA, 2002.

[6] *http://www.snort.org*.

[7] Sergei et al. *"SNORTRAN: An Optimizing Compiler for Snort Rules"*. Fidelis Security Systems, Inc., 2002.

[8] *http://www.xilinx.com*.

[9] John. W. Lockwood. *"An Open Platform for Development of Network Processing Modules in Reconfigurable Hardware"*. IEC DesignCon, Santa Clara, CA, 2001.