Exploiting Reconfigurable Hardware for Network Security

Shaomeng Li, Jim Torresen, Oddvar Soraasen
Department of Informatics, University of Oslo, N-0316 Oslo, Norway

Abstract— One type of network security strategy is using an Intrusion detection system (IDS). We are implementing an IDS in FPGA-based (Field Programmable Gate Array) reconfigurable hardware. This is to achieve higher speed and more efficient performance of network security, as networks develop very fast with consequently more demanding constraints. This study provides novel hardware architectures for an IDS system which should be able to monitor networks with a speed up to 2.68 Gbps.

1 Introduction

In this paper we examine the role of using dynamically reconfigurable logic (FPGA) to the network security design. FPGA-based reconfigurable hardware can be programmed almost like software, maintaining the most attractive advantage of flexibility with less cost than traditional hardware implementations. String matching is a key to achieving good performance in network security strategies. We study an intrusion-based string matching implemented in hardware and consider reconfigurable implementations based on FPGAs. Therefore this approach can be faster than software-based techniques and less expensive than some attractive ASIC (Application Specific Integrated Circuit) based hardware techniques. Intrusion Detection System and the reconfigurable hardware technology are described in section 2. The paper continues to present novel network security architectures based on reconfigurable logic in section 3. Our experiments will be shown in section 4. We conclude the paper in section 5.

2 Description

Intrusion detection systems would help the system designer to prepare for avoiding and dealing with attacks. IDSs generally accomplish this goal by collecting informations from a variety of system and network sources to build security rulesets to be used for matching against incoming packets. A survey of IDS can be found in [1].

Recently, the idea of using reconfigurable resources along with a conventional processor has lead to research in the area of reconfigurable computing. The main goal is to take advantage of the capabilities and features of both resources. While the processor deals with all the general-purpose computations, the reconfigurable hardware acts as a specialized coprocessor that deals with specialized operations where the parallelism, regularity of computation, like repeated comparisons of strings, can be exploited by creating custom operators, pipelines, and interconnection pathways. By allowing the programmer to change the hardware of the machine to match the task at hand, computational efficiency can be increased and an improvement in performance can be realized. More can be found in [4].

3 Exploiting New Method for IDS

Snort [2] is a lightweight open source network intrusion detection system, capable of performing real time traffic analysis and packet logging on IP networks.

In this paper, the detection engine of Snort in intrusion detection mode will be analyzed to speed up the Snort implementation as the detection engine took a considerable part of the Snort CPU time.

Snort maintains its detection rules in two dimensional rules lists. These are Rule Tree Nodes (RTNs) and Option Tree Nodes (OTNs). The IDS Snort stores detection informations such as the source and destination addresses, source and destination ports and protocol type (TCP, ICMP, UDP) into RTNs and TCP flags, ICMP codes and individual detection patterns etc into OTNs to form the rulesets.

In Snort v.1.8.7, with 127 RTNs and 1239 OTNs, there are 1239 different rules to value against incoming packets. Through ParseRuleFiles, all rules are translated into a data structure as seen in Table 1 and as discussed in [7].

Protocol	# RTN	#OTN	Max.	Min.
IP	4	32	19	1
ICMP	4	33	27	1
UDP	31	82	23	1
TCP	88	1092	728	1

Table 1. Snort v.1.8.7 Data Structure

We propose a detection engine implementation for this rulesets structure in reconfigurable hardware as seen in Figure 1.

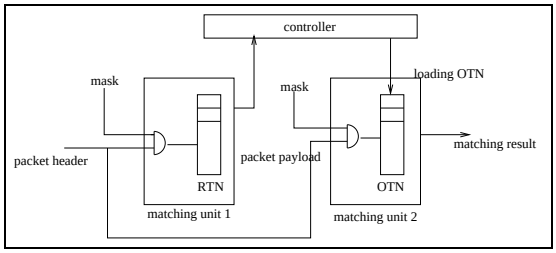

Figure 1. Reconfigurable datapaths for IDS Snort v. 1.8.7

Matching 1 conducts **header** matching of a packet upon all RTN rulesets and Matching 2 conducts further **payload** matching of the packet upon the corresponding OTNs of the matched RTN. After the matching 1, the controller will analyze which rulesets in RTNs that had a match and load the corresponding OTNs for the specific RTN into OTN hardware for matching 2. Therefore the speed is accelerated both by hardware parallel implementation and by avoiding a lot of comparisons at matching 2 as a packet is matched just to one OTN set, not all sets of OTNs. In [6], on the other hand, all patterns of rules have to be compared upon incoming packets.

In matching unit 1, RTN hardware stores the complete RTN rulesets when it is configured. In matching unit 2, OTN hardware is loaded with the corresponding OTN rules of the dedicated RTN which has given a match in matching unit 1. Matching units (matching 1 and matching 2) operate in parallel by pipelining the incoming packets, therefore execute separately RTN and OTN string matching on FPGA-based hardware.

4 Experiments

The complete matching unit is to be implemented using a CAM (Content Addressable Memory) as proposed by Xilinx [3] which has a high matching speed of a single clock cycle.

When a packet to be collected by packet capture engine (libpcap) arrives, the header of the packet is matched upon RTN rules. The matching result will be controlled to decide which OTN rules to load into the CAM in the OTN implementing hardware. By pipelining, the RTN implementation keeps on to match the next packet while OTN rules are loaded and matched in the matching unit 2.

Take the Snort v.1.8.7 for example: with 127 RTNs, all bits of IP, Protocol and Port of destination and resource hosts are 104 bits. 127 entries (word depth) and 104 bits matching data (word width) CAM will ideally be needed. In Xilinx, the SRL16E built CAM is fully parameterized by word depth and word width, however CAM word depth limits to a multiple of 16 words value and the min. word depth is

32 words. A 128 word depth x 104 bits word width CAM is therefore needed to implement the 127 RTN rules in Snort v.1.8.7. The max. number of OTNs is 728 according to Table 1. For checking the payload of packets against the OTN rules, max. size 768 word depth x 160 bits word width CAM, by considering limitations of word depth of CAM, will be needed. This is a worst case, and the best case will just need 32 word depth x 160 bits word width CAM for implementing the one OTN rule as the min. number of OTN is one according to Table 1. The pattern matching size is supposed to be up to 160 bits.

The size of RTN is fixed, and the size of OTN varies, therefore the built-in Xilinx CAM is suggested to implement the RTN matching unit. A different CAM than Xilinx CAM should be designed to implement the OTN matching unit. However the block diagram of implementing both can basically be shown in Figure 2. We detail this in the next sections.

4.1 RTN implementation

A total of 104 bits are required and stored in one 104 bit register for further CAM implementation against the RTN rules.

There are three ways to implement CAM designs in Xilinx Virtex devices: distributed SelectRAM based, Block SelectRAM based and built-in shift register (SRL16E) based.

The CAM built from the SRL16E which then needs only one clock cycle for read and 16 clock cycles for write, is used in the RTN implementation. Concerning the size of a CAM, a 32-word by 16-bit CAM would require 128 LUTs, a 128-word by 104-bit CAM would require 832 CLBs which could be implemented in XCV 150. Our implementation use the ISE FPGA tool from Xilinx. Our RTN implementation design is mapped to the XCV1000-6 FPGA which has 64x96 CLBs, and the worst delay time is 13.615 ns. So the design can be run at 73 MHz for each 32 bit dataword.

Just one 104 bit register is considered in the RTN hardware and other RAMs/buffer are not needed in the RTN implementation. One reason would be to save the critical resource usage of FPGA because the RTN implementation doesn't result in the bottleneck of matching the whole packet. Even if the speed of RTN implementation could be accelerated by buffering header information of the incoming packet for RTN computation, however the bigger bottleneck of matching the whole packet resides in the OTN implementation.

4.2 Loading OTNs of the corresponding RTN

The CAM built from the Virtex Block SelectRAM needs just a single clock cycle both for read and write. However, the maximum CAM size configured from Block SelectRAM has a limit at 4096 bits in XCV1000 which is already a considerable size of an FPGA device. Even though it is ideal for our system implementation with many OTN loadings to use CAM built from Block SelectRAM, our OTN design implemented by Block SelectRAM will not be an alternative. This is because the CAM size is restricted to 4096 bits and our design needs a larger number of bits if the loading of OTNs is implemented by using one clock cycle.

The SRL16E built CAM is still used to implement the OTN with 16 write clock cycles. However the loading time could ideally be omitted as the 16 clock cycles needed for loading OTNs into CAM is very small compared to the time needed for OTNs matching upon the rather large payload of the packet.

4.3 OTN implementation

The best and worst case is considered separately with CAM[1] 32 x 160 and CAM 768 x 160 distributed among three pieces of CAM 256 x 160. The matching pattern size could be up to 160 bits.

For accelerating the OTN performance, the OTN buffer is needed. When the header of a packet has a match against a RTN rule on the RTN hardware, the corresponding OTNs will have to be loaded onto the CAM of the OTN hardware. Pipelined RTN hardware matches the header of next packet while the OTN hardware is conducting their

[1]CAM is SRL16E CAM from Xilinx by some modifications

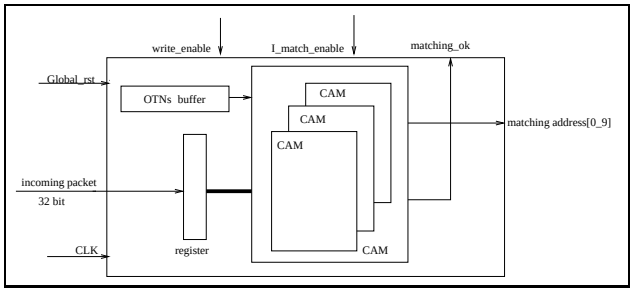

Figure 2. OTN/RTN matching details

content matching against the payload of the first packet. The OTN buffer is therefore needed for storing the dedicated OTN rulesets of the next packet determined by the result of the RTN processing of this packet in the RTN matching unit.

The feature of Virtex FPGA XCV1000 having 131,072 bits built-in dual-port block SelectRAM will be ideal for implementing the 3 x 256 x 160 bits OTNs buffer.

We mapped the design without the OTN buffer onto FPGA XCV1000-6. The result shows that the system could be run at 84 MHz for each 32 bit datapath, the system could therefore filter a 2.68 Gbps network. Even better results by applying an OTN buffer into the system are expected.

The RTN and OTN implementations execute in parallel, checking the header and payload of the packet respectively. However, the RTN implementation is matching the fixed fields of the packet header and the OTN implementation is using a sliding window to match the large payload of the packet. The sliding window has the fixed size of 160 bits, shifting in a 32 bit word of the packet each clock cycle by ignoring the time of loading the corresponding OTNs.

As the size of the payload of the packet is often large, the content pattern matching (OTN implementation) of the rules against the payload of the packet will be the main work of the complete system. The results of the OTN implementation performance would give an IDS performance of 2.68 Gbps.

5 Conclusions

In this study, we have focused on delegating comparison operations to reconfigurable hardware for accelerating a network security implementation. By being different from other approaches as discussed in [6], all RTNs are implemented in parallel in the RTN matching unit, by loading just one set of the corresponding OTN in the OTN matching unit where the content of the rules is compared to the incoming packet payload. We mapped the RTN and OTN matching unit separately into XCV1000 FPGA. The result is that the IDS performance obtained is better than 2.68 Gbps without doing any optimizing efforts, although there is still some work needed to be done to the system in the future regarding the omitted time for loading dedicated OTNs onto the OTN matching unit. In addition to the bottleneck described above, there are still more bottlenecks that should be avoided to implement high speed IDS. These include decode engines, some preprocessors and the libpcap packet capture, as discussed in [5].

References

[1] *http://www.sans.org/newlook/resources/IDFAQ/.*
[2] *http://www.snort.org.*
[3] *Xilinx CAM Application Note.*
[4] A. DeHon. *"Reconfigruable Architecture for General-Purpose Computing"*. MASSACHUSETTS INSTITUTE OF TECHNOLOGY, ARTIFICAL INTELLEGENCE LABORATORY, Technical Report.
[5] N. Desai. *"Increasing Performance in High Speed NIDS"*. 2002.
[6] M. Gokhale et al. *"Granidt: Towards Gigabit Rate Network Intrusion Detection Technology"*. in Proc. of 12th International Conference, FPL2002, France, Los Alamos National Laboratory, 2002.
[7] C. Kruegel et al. *"Automatic Rule Clustering for improved, signature based Intrusion Detection"*. University of California, Santa Barbara, 2002.