Pattern Based Packet Filtering using NetFPGA in DETER Infrastructure

Andrew Goodney, Shailesh Narayan, Vivek Bhandwalkar, Young H. Cho
Information Sciences Institute
University of Southern California
Marina Del Rey, CA 90292
{goodney, shailesn, bhandwal,younghch}@usc.edu

ABSTRACT

The Cyber DEfense Technology Experimental Research (DETER) testbed is a networking testbed that allows researchers to perform security focused simulation and experiments in a controlled environment. In this paper, we describe the integration and the use of a hardware/software co-design host with the NetFPGA card, an open source field programmable gate array (FPGA) based network interface. Through our case study, we also demonstrate how schematic based module design can simplify development of NetFPGA modules. The case study module is a simplified network intrusion detection system which uses deep packet inspection. We deploy and exercise our system using the DETER testbed.

1. INTRODUCTION

Internet worms and viruses account for billions of dollars in economic damage every year. Various network attacks infect and spread through custom designed packet payloads and network traffics. One effective way of detecting and preventing network attacks is by the way of deep packet inspection. Deep packet inspection not only examines headers but also the payloads of packets.[23] Therefore, a security system that incorporates a deep packet filter offers better protection from attacks than traditional firewalls.

However, scanning the payload of every packet at every byte requires high computation requirement; especially if there are multiple patterns that needs to be matched. Using software based approaches, detecting a reasonable set of string patterns in network packet over 1Gbps is a difficult task even on the latest general purpose multiprocessors.[26] For the past several years, a number of researchers have investigated novel ways to implement and accelerate the pattern matching tasks on field programmable gate arrays (FPGA). Many designs match tens of thousands of patterns at performances well beyond the practical limits of software based systems.

The NetFPGA network interface card is a platform that can be used to develop and test FPGA based algorithms for deep packet inspection. However, directly writing Verilog or VHDL code may not be the easiest design paradigm, especially for novice hardware designers. In this paper we describe a simple module for deep packet inspection designed completely using a graphical, schematic design paradigm. We show how this code can be successfully integrated with the reference code (written in Verilog) provided by the NetFPGA project.

Researchers not only need an easy-to-use hardware prototyping environment, but they also require a testbed in which to perform realistic experiments that will rigorously test a hardware design. The DETER testbed is designed for network security focused experimentation and simulation. Deploying NetFPGA in DETER is a natural fit and provides a hardware/software researcher with an environment where experiments with various desirable conditions (i.e. high-bandwith, high-packet rates or packets with dangerous or malicious content) can be created and carried out in a controlled environment.

Combining these technologies we have developed a high-speed pattern matching module for the NetFPGA using schematic design. We deployed and tested the design in DETER to both validate our hardware design, but to also prove the suitability of DETER for such experiments. This paper presents a basic hardware accelerated network intrusion/prevention detection system (NIDS/NIPS) for NetFPGA platform. In section 3, the paper shows how a parallel pattern matching engine and specialized first-in first-out (FIFO) module are integrated to the reference gigabit router. The implementation and experimental details are discussed in section 4. Then in section 2, the some of the relevant prior works are briefly discussed to suggest other potential extensions to the presented system. The paper concludes in section 5 with a few final thoughts on the future of network security systems on reconfigurable platforms.

2. RELATED WORKS

2.1 NetFPGA

To date, FPGA based network processors have often been custom hardware designed for the task at hand, thus limiting other researchers ability to duplicate and enhance designs. The NetFPGA [22] platform contains many of the necessary components to build FPGA based network processors on an open source, commercially available PC board.

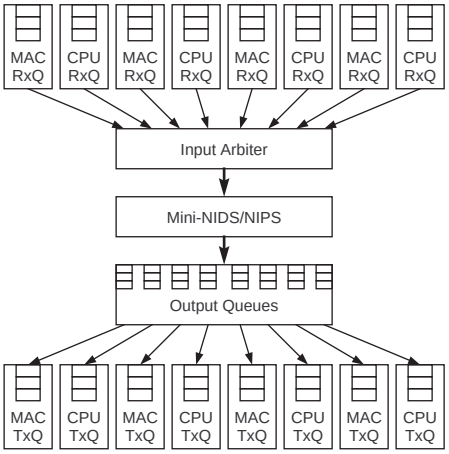

Figure 1: Mini-NIDS/NIPS system within the reference router design

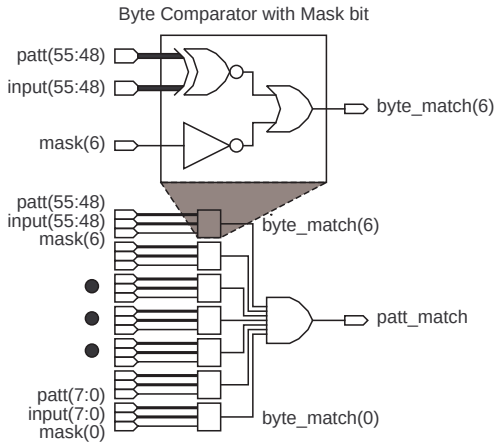

Figure 2: Parallel pattern matching engine

NetFPGA allows researchers and students to build and test network systems using two widely available technologies: Gigabit Ethernet and Xilinx FPGAs. The NetFPGA expansion card consists of Four Gigabit Ethernet ports, Xilinx FPGA, Static RAM (SRAM), Double-Data Rate RAM (DRAM) and PCI host interface. The board can be installed a computer to make the system act as any type of router, Firewall, IDS/IPS or a Measurement device. The NetFPGA developers provide a set of base packages and reference code that can be used to configure the NetFPGA as a 4-port network interface card, or 4-port router. The reference code handles packet I/O to the host CPU and Ethernet interfaces and since it is modular can be used as a starting point for module design.

The modular design combined with Gigabit Ethernet interfaces provide an easy to use environment for designing and testing high-speed, hardware based network processing systems or algorithms. Other researchers have built systems on the NetFPGA. These systems have implemented technologies such as layer 2 virtualization, routing protocols, switching, URL extraction, packet generation and general purpose programable packet processing. See [1, 20, 27, 21, 6, 24, 8, 16, 13, 25, 30] for work of this type. This body of NetFPGA work shows the flexibility and power of the platform.

2.2 DETER

In last couple of decades, computers have penetrated in our lives substantially. Today, many sectors, such as Finance, Medical, Education, Defense, Entertainment, etc rely upon computers to perform majority of their tasks. While this has made our lives easier, it has also exposed us to certain risks such as Cyber attacks. In their nascent stage, cyber attacks were not able to do much harm, but now attacks have become lot more organized and sophisticated which leaves us even more vulnerable to such threats and malicious activities. We need to guard ourselves along with our infrastructure from these attacks.

Research has been done to simulate/emulate such attacks and study their behavior [12, 17]. Many companies and research organizations built testbeds to study certain kinds of attack, but these testbeds suffered two serious drawbacks: 1. Collaboration was impossible outside the research group; 2. Testbeds were attack specific and non-adaptable. There was a need for a general purpose testbed which is not only flexible but also allows collaboration among security researchers from different organizations and nationalities.

The Cyber Defense Technology Experimental Research network (DETER) testbed [2] is a combination of physical network infrastructure, experimental methodologies and management tools. The testbed is fully isolated from the Internet and all the experiments execute in a controlled environment. Experiments are encouraged to generate malicious traffic and may cause damage to host computers or other network infrastructure. The DETER management tools are able to clean and reset the computers, switches and routers back to a known good state. The utility of DETER will depend on the power, convenience, and flexibility of its software for setting up and managing experiments including registration, definition, generation control, monitoring, check-pointing, and archiving. An important aspect of the management software is the requirement for sophisticated network monitoring and traffic analysis tools for both experimenters and DETER network operators. Experimental results obtained using the DETER testbed enables researchers to develop fast, efficicent and innovative technologies which could be used widely providing cyber protection.

DETER is hosted at USC/ISI and University of California, Berkley. It is based on the Emulab [14] project at University of Utah. Using DETER, researchers set up experiments containing PCs, switches, and routers which can be configured into arbitrary topologies. Experiments are then carried out in a controlled manner. As a shared resource, experiments can be larger and more sophisticated than what a researcher may be able to afford if he were required to acquire and configure the necessary equipment himself.

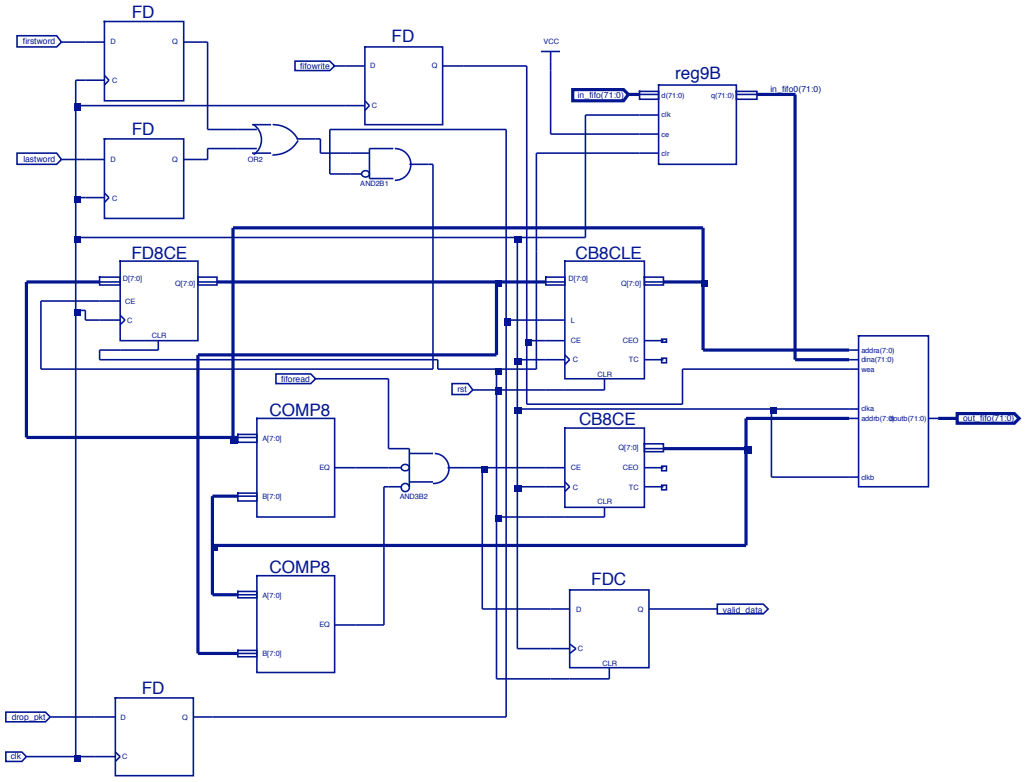

Figure 3: The screen capture of Xilinx ISE Schematic Layout Tool of the drop enabled FIFO module

2.3 FPGA Based Intrusion Detection

There are a number of fast pattern search algorithms implemented in FPGA to meet the performance requirements of high-bandwidth NIDS/NIPS. Sidhu and Prasanna mapped Non-deterministic Finite Automata (NFA) for regular expression into FPGA to perform fast pattern matching [28]. Using this method, Franklin et. al compiled patterns for an open-source NIDS system into JHDL [10]. Work from Washington University in St. Louis shows that NFA translated into deterministic finite automata (DFA), in practice, optimizes to compact and fast hardware for the purpose of NIDS[18]. Due to the parallel nature of the hardware, these designs maintained high performance regardless of the size of the patterns.

The Granidt project of Los Alamos National Laboratory implemented a fast re-programmable deep packet filter using content addressable memories (CAM) and the Snort rule set [11]. UCLA's implementation of the same application uses discrete logic to build fast pattern match engines. The match engine is a parallel pipelined series of reconfigurable lookup tables (LUT) that can sustain a bandwidth of 2.8 Gbps [5][4][19]. Sourdis mapped a similar design with a deeper pipeline to increase the filtering rate [29] while Clark made some area improvements [7].

Hash function based systems have also been implemented on FPGA based system. One notable system uses Bloom filters [3] to detect the entire set of Snort signatures at line rate.[9, 15].

Based on some of the earlier exact pattern matching design, we implement and integrate a on-line programmable pattern matching module to NetFPGA reference router. We describe this work in more detail in the following section.

3. INTRUSION DETECTION/PREVENTION SYSTEM

3.1 Extended the Reference Router Design

To develop a module for the NetFPGA a designer has two options. One may write the module in a high-level hardware language such as Verilog, or one may use a graphical schematic capture tool. We posit that for many NetFPGA implementations the schematic design option is advantageous. For example, if the NetFPGA is being used by students in a graduate computer science or electrical engineering course, the students may have various amounts of prior hardware experience. However, they will all have seen block-level hardware or computer architecture designs. By using hardware modules (memory, registers, flip-flops, etc.) provided by the standard library for Xilinx FPGAs, designers can design and simulate complex hardware designs without learning a new programming language.

In order to implement the NIDS idea described above we extended the reference router design of the NetFPGA. The reference router design configures the NetFPGA as a 4-port router. The interface addresses and routing information are configured at runtime from the host. Packets are then ex-

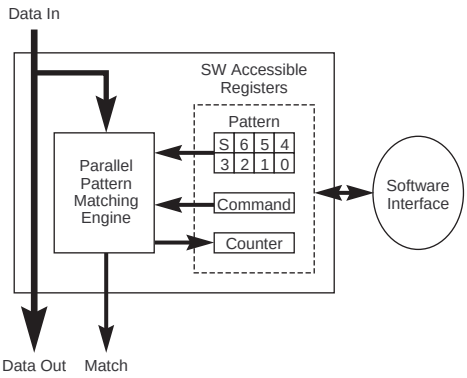

Figure 4: Block diagram of mini-NIDS/DPI system

amined and routed solely inside the FPGA hardware. This allows for full-speed, full-duplex routing on each port, totaling 8GBps of throughput.

The reference router has a set of I/O queues for each port. Received packets are placed in the input queues and then examined. If the forwarding information for a packet is available in the routing table, the packet is placed in the proper output queue. Between the output of the routing engine and the output queues it is easy to insert a module that sees all packets passing through the router. It is here that we insert our NIDS module. The module is based on a fixed length FIFO, thus it has no effect on the routing throughput of the router, while enabling the NIDS functionality.

Using the Xiling ISE integrated design environment we first created a project that imported the Verilog source code for the reference router. We then have access to the data and signaling lines using the labels contained in the Verilog source. We used block-level modules available from the Xilinx component library such as dual-ported SRAM, counters, comparators, and registers to construct the pattern matching logic and drop enabled FIFO. Before compiling the design to a bitfile we used the simulation environment to debug the design.

3.2 Drop Enabled FIFO

The drop enable FIFO is used to store packets as they are scanned. The internal bus of the reference router is 8 bytes wide, so to ease integration with the reference router the drop-enabled FIFO is also 8 bytes wide. It is constructed of dual-ported SRAM and is approximately 3KB total in size (enough to hold two standard size Ethernet packets).

Bytes from a packet are written into the FIFO with an address counter and simultaneously scanned by the string matching engine. If a packet does not contain a string match, a second counter is used to write it out to the next module (usually the output arbiter).

If a packet contains a matching string, the packet is dropped by simply resetting the address counter and writing over the packet with the next arriving packet.

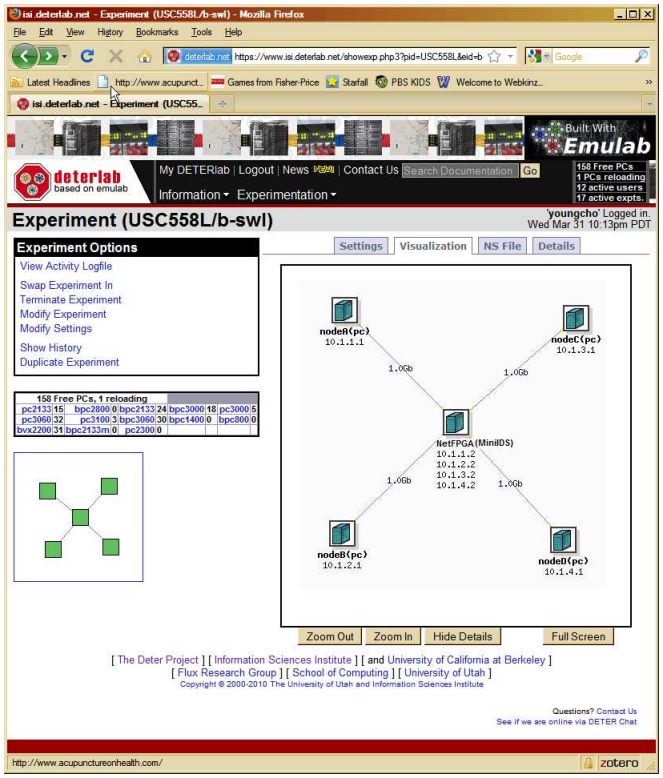

Figure 5: Screen shot of the NetFPGA integrated with DETER

3.3 String Pattern Matching Engine

The String Pattern Matching Engine is the core of the NIDS module. Figure 2 shows the design of the component. Bytes in the FIFO are compared against the 7-byte string for a match. The matching logic is repeated in such a way that as each 8 byte section of the packet arrives into the FIFO, all possible byte alignments are scanned. If a match is detected, the packet is dropped as described above.

3.4 Software Interface

In addition to the FPGA hardware design of the NIDS, we provide a simple software interface. This software interface allows the user to configure the string pattern against which the hardware matches. The software interface also allows to read and reset the counter incremented by the matching engine. Finally, a command interface is implemented so other run-time aspects of the hardware design may be modified (i.e. changing from count-only to drop modes).

4. IMPLEMENTATION AND RESULT

4.1 Integration with DETER

DETERLab gives us the flexibility (for creating different test environments) and isolation (so that we can run malicious code and observe their behavior without running into risk of releasing it to other parts of the Internet). NetFPGA gives us the flexibility (of changing the hardware design through FPGA re-programming) and hardware speed. Its a perfect blend if we combine these two technologies which can help us in coming up with faster security solutions.

(a) Bandwidth with NetFPGA + reference router

	Node A	Node B	Node C	Node D	Total Out B/W
Node A	-	262	287	285	834
Node B	281	-	258	286	825
Node C	292	263	-	259	814
Node D	257	297	284	-	838
Total IN B/W	830	822	829	830	

(b) Bandwidth with NetFPGA + (reference router & NIDS)

	Node A	Node B	Node C	Node D	Total Out B/W
Node A	-	301	286	273	860
Node B	273	-	273	268	814
Node C	237	280	-	298	815
Node D	289	258	285	-	832
Total IN B/W	799	839	844	839	

(c) Bandwidth with NetFPGA + (reference router & NIDS dropping packets)

	Node A	Node B	Node C	Node D	Total Out B/W
Node A	-	0	414	437	851
Node B	432	-	0	401	833
Node C	385	433	-	0	818
Node D	0	397	414	-	811
Total IN B/W	817	830	828	838	

Table 1: Experimental results of NetFPGA based NIDS deployed on DETER

We used a DELL 150 server and academic version of the NetFPGA card. The integration process involved two steps: 1) We installed the NetFPGA card on the machine, installed all the drivers and base packages required by the NetFPGA card for normal working. Then we tested it thoroughly. 2) Finally, we put the machines into the DETERLab and integrated it into the bigger system. Then using Deter tools, we were able to load CentOS 5.4 on it and install all the base packages and required drivers. The last step was to take a custom image, so that future users can directly load the custom image to the NetFPGA node and start experimenting.

4.2 Using NetFPGA in DETER

For users familiar with DETER (or Emulab) this section outlines how to use the NetFPGA in DETERlab. The DETERlab testbed uses the Emulab cluster testbed software developed by the University of Utah. This software controls a pool of experimental nodes that can be assigned, interconnected with high-speed links by any type of topologies, loaded, and monitored remotely, to meet the requirements of each experiment. Researchers or experimenters use the DETERLab web interface to create, swap-in, swap-out, and monitor their experiments remotely.

One of the important functionality Deter provides is its ability to provide interface to design and implement different types of topologies. This is achieved by using configuration which describes the network topologies. Just as Emulab, DeterLab uses the "NS" (Network Simulator) format to describe network topologies.

We have integrated the NetFPGA into DETERLab in a manner that is compatible with the NS file format. To include the NetFPGA in their experiment requires only 5 additional lines. Users must declare the OS-type for one node to be `CentOS5-netfpga-base`. The DETERLab system administrators have marked only nodes containing a NetFPGA card as capable of running this OS image, resulting in a NetFPGA node being assigned to your experiment. Secondly, users must declare 4 nodes of type `netfpga2`. The individual interfaces on the NetFPGA are seen as virtual nodes in DETER, so one must declare them in this manner in order to map them into the experimental network. Please see listing 1 for an example NetFPGA NS file.

Users must have their own workstation properly licensed with the Xilinx tool-chain in order to generate bitfiles for the NetFPGA. Once a NetFPGA experiment is running, users upload their bitfile to the DETER node running `CentOS5-netfpga-base`. From there the bitfile is loaded to the NetFGPA and experimentation can begin. If the NetFPGA is configured with the reference NIC or reference router code, the NetFPGA host machine will have 4 Ethernet ports available to the Linux OS representing the 4 Host CPU queues implemented by the output arbiter.

4.3 Performance

The reference router NetFPGA design is such that it allows wire-speed packet routing on full-duplex 1Gbps ports. As described above, our NIDS is also designed to be inserted into the reference router pipeline. Since packets are examined in the FIFO as they pass through our NIDS, our addition to the reference router should have no impact on the wire-speed routing capabilities of the NetFPGA. We have designed an experiment to test this assumption and to show that our NIDS performs as designed.

Using DETER we configured a network with the topology shown in figure 5 (also listing 1. Each host was connected through a gigabit switch to a port on the NetFPGA board. First we ran a simple experiment to show that we do not place any additional constraints on the routing performance. Here we load the reference router bit file as supplied by the NetFPGA project. We run a TCP Iperf server on each of the

4 non-NetFPGA nodes. Then we run 3 TCP Iperf clients on each of the nodes. Each client connects to one of the other 3 nodes. The clients attempt to send data at maximum bandwidth. Therefore the bi-directional bandwidth seen at each NetFPGA port should approach 2Gbps. Table 1(a) shows the matrix of throughput values obtained.

Secondly we perform the same experiment, however this time we load a bit file that includes our NIDS module. The NIDS module is not configured to drop traffic. As can be seen in table 1(b) the throughput obtained by the individual nodes and the system as a whole is unchanged.

Third we verify that the NIDS properly blocks traffic that contains the selected pattern (we call this BAD traffic). One TCP Iperf client on each node is configured to send packets that contain data that matches the selected pattern. The other Iperf clients operate as before. In table 1(c) we see that not only do the BAD TCP connections fail to achieve any throughput, the other TCP connections are able to exploit the extra bandwidth. The overall system throughput remains the same. Finally we ran an experiment where all Iperf clients were configured to send BAD traffic. As expected, the system experienced zero throughput.

5. CONCLUSION AND FUTURE WORK

Many researchers are interested in how reconfigurable hardware might be used to develop novel network processing hardware/software systems, particularly as related to cyber security. The NetFPGA network interface card provides an easy to use platform for designing FPGA based network hardware, while the DETER testbed provides a flexible arena to perform repeatable, medium-scale network security experiments. In this paper we show how schematic design can speed the NetFPGA hardware design process by designing deep-packet inspection engine. We then deploy the design on DETER and experimentally verify the performance.

Currently (Spring 2010) the NetFPGA deployed on the DETER testbed was under beta-test by the students in a graduate-level network systems course at the University of Southern California. We plan to make the NetFPGA available to all DETER researchers over Summer 2010. We will release detailed instructions on how to allocate, configure, and experiment with NetFPGA on DETER. In the future we will investigate extensions to the DETER interface (web and command-line) that will allow seamless integration, such as automatic generation and upload of a bitfile from Verilog source.

The authors would like to thank John Hickey and Ted Faber of the DETER project for their invaluable help in deploying the NetFPGA node on DETER.

6. REFERENCES

[1] M. B. Anwer and N. Feamster. Building a fast, virtualized data plane with programmable hardware. In *VISA '09: Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pages 1–8, New York, NY, USA, 2009. ACM.

[2] T. Benzel, R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab. Design, deployment, and use of the deter testbed. In *DETER: Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test 2007*, pages 1–1, Berkeley, CA, USA, 2007. USENIX Association.

[3] B. H. Bloom. Space/Time Trade-Offs in Hash Coding with Allowable Errors. In *Communications of the ACM*. ACM, July 1970.

[4] Y. H. Cho and W. H. Mangione-Smith. A pattern matching coprocessor for network security. In *Proceedings of the 42nd annual conference on Design automation*, pages 234–239, 2005.

[5] Y. H. Cho, S. Navab, and W. H. Mangione-Smith. Specialized Hardware for Deep Network Packet Filtering. In *12th Conference on Field Programmable Logic and Applications*, pages 452–461, Montpellier, France, September 2002. Springer-Verlag.

[6] M. Ciesla, V. Sivaraman, and A. Seneviratne. URL Extraction on the NetFPGA Reference Router, 2009.

[7] C. R. Clark and D. E. Schimmel. Scalable Parallel Pattern-Matching on High-Speed Networks. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, April 2004. IEEE.

[8] G. A. Covington, G. Gibb, J. W. Lockwood, and N. Mckeown. A packet generator on the netfpga platform. *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on*, 0:235–238, 2009.

[9] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood. Deep Packet Inspection using Parallel Bloom Filters. In *IEEE Hot Interconnects 12*, Stanford, CA, August 2003. IEEE Computer Society Press.

[10] R. Franklin, D. Carver, and B. L. Hutchings. Assisting Network Intrusion Detection with Reconfigurable Hardware. In *IEEE Symposium on Field-programmable Custom Computing Machines*, Napa Valley, CA, April 2002. IEEE.

[11] M. Gokhale, D. Dubois, A. Dubois, M. Boorman, S. Poole, and V. Hogsett. Granidt: Towards Gigabit Rate Network Intrusion Detection Technology. In *12th Conference on Field Programmable Logic and Applications*, pages 404–413, Montpellier, France, September 2002. Springer-Verlag.

[12] G. Kesidis, I. Hamadeh, Y. Jin, S. Jiwasurat, and M. Vojnovic. A model of the spread of randomly scanning internet worms that saturate access links. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 18(2), 2008.

[13] M. Labrecque, J. G. Steffan, G. Salmon, M. Ghobadi, and Y. Ganjali. NetThreads: Programming NetFPGA with Threaded Software, 2009.

[14] W. D. Laverell, Z. Fei, and J. N. Griffioen. Isn't it time you had an emulab? In *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 246–250, New York, NY, USA, 2008. ACM.

[15] J. Lockwood, J. Moscola, M. Kulig, D. Reddick, and T. Brooks. Internet Worm and Virus Protection in Dynamically Reconfigurable Hardware. In *Military and Aerospace Programmable Logic Device (MAPLD)*, Washington DC, September 2003. NASA Office of Logic Design.

[16] J. Luo, Y. Lu, and B. Prabhakar. Prototyping Counter Braids on NetFPGA, 2008.

[17] J. Mirkovic, A. Hussain, S. Fahmy, P. Reiher, and R. K. Thomas. Accurately measuring denial of service in simulation and testbed experiments. *IEEE Transactions on Dependable and Secure Computing*, 6(2), 2009.

[18] J. Moscola, J. Lockwood, R. Loui, and M. Pachos. Implementation of a Content-Scanning Module for an Internet Firewall. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, April 2003. IEEE.

Listing 1: Example DETERLab NS file utilizing a NetFPGA node

```
# This is a simple ns script. Comments start with #.
# Below 2 lines are the prologue as mentioned above
# This script will setup 5 nodes in a star topology:
# 4 regular Linux nodes and one node to host the NetFPGA

set ns [new Simulator]
source tb_compat.tcl

#Then define the 5 nodes in the topology.

set control [$ns node]
set nf1 [$ns node]
set nf2 [$ns node]
set nf3 [$ns node]
set nf4 [$ns node]

set nodeA [$ns node]
set nodeB [$ns node]
set nodeC [$ns node]
set nodeD [$ns node]

#The following lines import the 4 ports on the NetFPGA
tb-set-hardware $nf1 netfpga2
tb-set-hardware $nf2 netfpga2
tb-set-hardware $nf3 netfpga2
tb-set-hardware $nf4 netfpga2

#The following line ensures the NetFPGA host is used
tb-set-node-os $control CentOS5-netfpga-base

tb-set-node-os $nodeA  CentOS5
tb-set-node-os $nodeB  CentOS5
tb-set-node-os $nodeC  CentOS5
tb-set-node-os $nodeD  CentOS5

set link0 [$ns duplex-link $nodeA $nf1 1000Mb 0ms DropTail]
set link1 [$ns duplex-link $nodeB $nf2 1000Mb 0ms DropTail]
set link2 [$ns duplex-link $nodeC $nf3 1000Mb 0ms DropTail]
set link3 [$ns duplex-link $nodeD $nf4 1000Mb 0ms DropTail]

$ns rtproto Static

$ns run
```


[19] J. Moscola, J. W. Lockwood, and Y. H. Cho. Reconfigurable content-based router using hardware-accelerated language parser. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 13(2):1–25, 2008.

[20] M. Motiwala, M. bin Tariq, B. Anwer, D. Andersen, and N. Feamster. A narrow waist for multipath routing. *School of Computer Science, Georgia Tech. Tech Report*, 2009.

[21] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown. Implementing an openflow switch on the netfpga platform. In *ANCS '08: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 1–9, New York, NY, USA, 2008. ACM.

[22] J. Naous, G. Gibb, S. Bolouki, and N. McKeown. Netfpga: reusable router architecture for experimental research. In *PRESTO '08: Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, pages 1–7, New York, NY, USA, 2008. ACM.

[23] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In *USENIX LISA 1999 conference*, http://www.snort.org/, November 1999. USENIX.

[24] E. Rubow. Packet Processing with PowerPC on the NetFPGA, 2009.

[25] G. Salmon, M. Ghobadi, M. Labrecque, and J. G. Y. Steffan. NetFPGA-based Precise Traffic Generation, 2009.

[26] D. L. Schuff. High-performance network intrusion detection through parallelism. Master's thesis, Purdue University, 2007.

[27] J. Shafer, M. Foss, S. Rixner, and A. L. Cox. The Axon Network Device: Prototyping with NetFPGA, 2009.

[28] R. Sidhu and V. K. Prasanna. Fast Regular Expression Matching using FPGAs. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, April 2001. IEEE.

[29] I. Sourdis and D. Pnevmatikatos. Fast, Large-Scale String Match for a 10Gbps FPGA-based Network Intrusion Detection System. In *13th Conference on Field Programmable Logic and Applications*, Lisbon, Portugal, September 2003. Springer-Verlag.

[30] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford. Virtual routers on the move: live router migration as a network-management primitive. In *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 231–242, New York, NY, USA, 2008. ACM.