A Combined Hardware-Software Architecture for Network Flow Analysis

S. Yusuf, W. Luk, M. Sloman, N. Dulay and E.C. Lupu
Department of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ, England
G. Brown
Department of Computer Science, Indiana University, Bloomington IN 47405, USA

Abstract. *It is increasingly difficult for network devices to keep pace with rapid developments in network data rate speeds. Many such devices are unable to match the OC-192 link speed. This paper describes the use of a combined hardware-software system as an application-specific solution to this problem. Our approach maps the performance-critical tasks of packet classification and flow monitoring from software into hardware using a field programmable gate array (FPGA), such that operations can run in parallel where desirable. A feature of our architecture is its capability to process multiple flows in parallel. We explore the scalability of our system showing that it can support flows at multi-gigabit rate, which is faster than most software-based systems where acceptable data rates are typically no more than 100 Mbps.*

1 Introduction

It is now common practice to monitor network traffic in order to track statistics on resource-utilisation to various destinations by specific protocols. This ability to monitor traffic by endpoints and protocols is a key ingredient in managing bandwidth utilisation and costs. To this end, enabling technology such as NetFlow [9] is often used. The NetFlow protocol is implemented by all major router vendors and provides "per flow" packet and byte counts.

Many existing monitoring systems are software based. However, they are unable to provide the required flow processing rate to keep up with potential network traffic rates (Table 1). For this reason most monitors employ packet sampling instead.

There has been comparable work carried out on employing reconfigurable hardware as a means of accelerating network processing speed [2, 4, 5, 7], but none focused on processing of network flows to obtain statistics that could be used to improve network performance.

Table 1: Throughput for different link speeds, with a packet size of 128 bytes. Mbps denotes million bits per second, while Kpps and Mpps denote thousand packets per second and million packets per second, respectively.

Link Speed	Max Packet throughput
10Mbps	10 Kpps
100Mbps	100 Kpps
1Gbps	1 Mpps
2.5Gbps (OC-48)	2.5 Mpps
10Gbps (OC-192)	10 Mpps
40Gbps (OC-768)	40 Mpps

The motivation for this paper is three-fold. Firstly, we aim to extend the existing NetFlow implementations to enable the collection of performance statistics with the intention of diagnosing performance problems, such as regions suffering from high bandwidth utilisation or packet loss. Secondly, we address the limitations of software implementations in processing traffic at current network rates. Finally, we provide a scalable hardware-based architecture, which can cover current and future speed levels of network devices.

Our approach maps performance-critical tasks of packet classification and flow monitoring into hardware, such that operations can run in parallel where desirable. Our proposed architecture is modular and enables the optimisation of individual modules independently of each other. In addition, we provide a simple analytical model of the system which can be used to estimate resource requirements based on the

desired performance, or to determine potential system performance based on available resources.

The contributions of this paper include: (1) a combined hardware-software monitor system, which produces flow statistics to identify problematic regions in the network based on a two-level architecture for traffic analysis (Section 3), (2) a technology independent model of the system which enables the analysis and prediction of the size and performance of the system (Section 4), (3) an initial traffic monitoring system which operates on a multi-gigabit network (Section 6), and (4) the use of run-time reconfiguration to provide flexibility for end-user requirements (Section 7).

2 Measurement Engine (ME)

The ME system is designed to monitor network traffic passively through multiple interfaces, and to produce NetFlow compatible data which can be either exported as UDP packets, or displayed through a Web-based graphical user interface. A basic assumption here is that all traffic to and from the Internet can be passively monitored. Estimating performance for a TCP connection requires the ability to "see" traffic in both directions. However, with multiple Internet connections in a network, there is little guarantee that traffic associated with both directions of a flow will utilise the same Internet connection. Thus, all packets must be linked to a particular flow already identified, or used to create a new flow.

Figure 1 is a block diagram of a complete ME NetFlow system, showing the primary components. The traffic monitored by an ME NetFlow collector is forwarded from edge routers through Ethernet mirror ports or optical splitters. Traffic received by the ME NetFlow collector at its interfaces is classified into flows based upon layer 2 and layer 3 addressing information (MAC addresses, IP address, protocol and port). The classified flows are then cached and periodically forwarded to the flow aggregator.

The main purpose of the ME is to collect statistics in order to diagnose performance problems such as determining destination regions suffering high bandwidth utilisation or packet loss, and also to determine whether the network's security has been breached in some way.

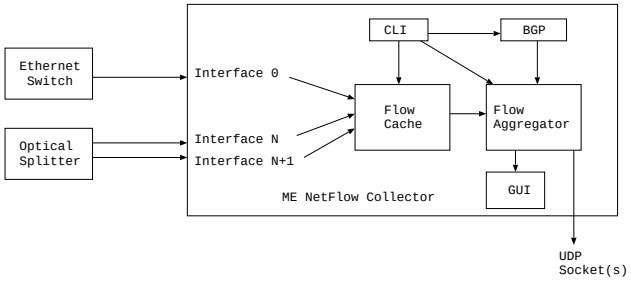

Figure 1: The ME NetFlow System. CLI and GUI respectively denote Command Line Interface and Graphical User Interface. The BGP block supports BGP routing.

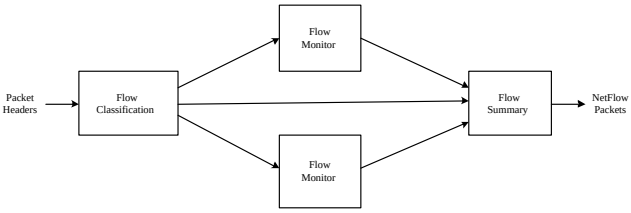

Figure 2: The Measurement Engine backend

As illustrated in Figure 1, the ME consists of a back-end, a flow cache which handles packets at wire rates, and a front-end which processes the resulting flow statistics at a much lower rate.

A view of the Flow Cache and Flow Aggregator as an ME back-end is illustrated in Figure 2. Packet headers are passed to the classifier to determine the flow to which the packet belongs. The classifier also decides which flow monitor is responsible for processing packets belonging to the flow. When a new flow arrives, the classifier allocates space for a "key" and assigns the flow to a flow monitor. The "key" to a flow consists of a five tuple, source and destination addresses, source and destination ports and protocol.

3 Two-Level Flow Analysis Architecture

Our approach partitions the system into a hardware-software system based on a two-level flow analysis scheme. This combined system allows the processing of flows at rates exceeding 1 Gbps.

The **hardware** element is responsible for processing flows, determining byte and packet count for flows, and returning statistics to the software. This is the first-level flow analysis which produces pre-

liminary statistics in the form of NetFlow-compatible packets, which are then further analysed by software.

The **software** element performs further analysis on the statistics derived from hardware. For example, take the scenario where all TCP packets to a server are counted, and compared to the normal average volume of traffic: the software detects that the volume of packets to the server is higher than normal. This may suggest suspicious activity that needs further investigation. Such events can alert network administrators without the need for human intervention [6]. In addition, inherent software flexibility also allows room for yet-to-be-defined user analysis criteria. We shall explain in Section 7 how run-time reconfigurability of the hardware elements can be exploited to support such user analysis ceriteria.

Figure 3 provides an overview of the system. It is possible for some of the software modules to be replaced by hardware modules described in Section 4 and Section 5. The resulting performance improvement is presented in Section 6.

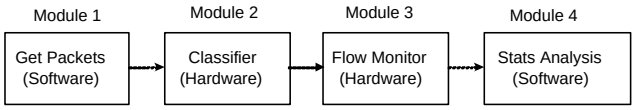

Figure 3: Hardware-Software system overview

Module 1: the current software implementation of the ME system receives traffic at its interfaces and buffers them for processing. When the buffer is full, or a specified time period has elapsed, the packets are transferred to subsequent modules. The use of software for online processing incurs a delay for buffering the packets and transferring them the classifier.

Modules 2 and 3 represent the first-level analysis and are detailed in Section 5.

Module 4: the software implementation receives the NetFlow-compatible packets and performs further analysis in order to determine areas of high packet loss or high bandwidth utilisation; this constitutes the second-level flow analysis. In Section 7, we shall explore how this analysis can be used to support recognition of suspicious activities and managed by run-time reconfiguration.

There are several possibilities for implementing the software blocks: (a) by employing the processor in a PC, (b) by utilising an embedded processor (for example a Power PC) on an FPGA (for example the Virtex II Pro), and (c) by making use of a soft core processor (for example the MicroBlaze).

To assess the feasibility of our system, we implement the software module of our prototype using a PC and, although not fully examined in this paper, the latter two approaches would allow us to eliminate the use of a slower PCI bus, since the processors are physically located on the FPGA.

A global view of the architecture of our hardware-based system is illustrated in Figure 4. In this architecture, each flow is allocated to a processing unit, with each flow monitor ultimately multiplexing over multiple flows. There are two possible methods of processing multiple flows: (1) we can assign each new flow to a flow monitor based upon queue length, allowing support of multiple memories, or (2) we can allow a packet to be processed by any flow monitor. With both methods of processing, there is a potential complication in that the packets associated with a single flow may be spread over several seconds or minutes; so with a possible 10,000 flows per second, there may be many times that number of active flows.

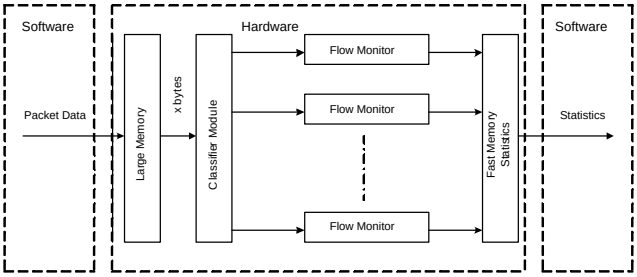

Figure 4: Architecture of the hardware components of the combined measurement engine

4 Technology-Independent Analysis

We quantify the size, N_{max}, of the maximum number of active flows that can be implemented in a reconfigurable device, by providing a technology-independent algebraic expression which can be used to estimate the system flow rate based on our architecture. For this purpose, we express N_{max} as a function of the small and fast internal memory space M_{small} and the number of logic cells L_{mon} required for implementing each flow monitor module. Also, we ex-

press the flow processing rate $R_{flow-rate}$ in terms of the number of N_{max} flows processed per second.

The maximum number of active flows that can be processed depends on the maximum amount of flow data, F_{data}, which can be stored in the small fast access memory.

To determine the number of flow processing engines that can be implemented in parallel, we assume that the modules depicted in Figure 4, except for the flow monitor modules, use a constant number of logic cells L_{const}; then we can express one possible maximum value of flow engines based on available reconfigurable logic, $N_{flow-engines}$, as:

$$N_{flow-engines} = (L_{total} - L_{const}) / L_{mon} \quad (1)$$

where L_{total} is the total number of logic cells available on the reconfigurable device. This value of $N_{flow-engines}$ is used if method (1), stated in Section 3, is used. However, with method (2), the maximum number of flow monitor modules that is implemented is derived by determining the number of pipeline stages required to obtain high throughput. Therefore, another possible maximum value of flow engines is:

$$N_{flow-engines} = C_{flow-cycles} \quad (2)$$

where $C_{flow-cycles}$ represents the number of cycles taken by one monitor engine to process one packet.

We calculate the number of flows which can be processed per second, $R_{flow-rate}$, in terms of the maximum operating frequency F_{max} of the system:

$$\begin{aligned} C_{N_{max}} &= (P_{ave} \times N_{max}) \times (P_{cycles} + C_{class-cycles}) \\ &\quad + C_{mon-cycles} + C_{cycles} \end{aligned} \quad (3)$$

$$R_{flow-rate} = (F_{max}/C_{N_{max}}) \times N_{max} \times P_{ave} \quad (4)$$

where $C_{N_{max}}$ is the total number of cycles to process N_{max} flows, $C_{class-cycles}$ is the number of cycles to classify N_{max} flows based on the average number of packets (P_{ave}), $C_{mon-cycles}$ is the number of cycles to process a flow consisting of P_{ave} packets, and C_{cycles} is the constant number of cycles needed during the processing of each flow.

In the calculation of $R_{flow-rate}$, we make use of $C_{N_{max}}$ which includes the time taken to transfer the required number of packets to classify N_{max} flows. We can calculate the packet transfer rate, $T_{pkt-trans}$, between the large memory M_{large} and the flow classifier. If x bytes can be transferred from M_{large} to the flow classifier per clock cycle, then

$$P_{cycles} = B_{packets}/x \quad (5)$$

where $B_{packets}$ is the number of bytes per packet and P_{cycles} denotes the number of cycles used to transfer a packet. Using Equation (5), we can express the packet transfer rate as:

$$T_{pkt-trans} = F_{max}/P_{cycles} \quad (6)$$

In Section 5 we utilise the equations to show how they are used to estimate the performance of the system, with Table 2 detailing the results.

5 Device-Specific Mapping

We develop a prototype for the proposed architecture targeting a development board which contains a Xilinx Virtex II XC2V8000 FPGA device and 6 external memory banks.

Module 1: Memory Transfer. We transfer the packet header from M_{large} to the flow classifier, at 4 bytes per clock cycle from each memory bank (6 in total). This uses approximately 2 clock cycles, P_{cycles} (Equation (5)), to transfer one packet header, achieving a $T_{pkt-trans}$ (Equation (6)), of up to 5 million packets per second at an approximate F_{max} of 10MHz.

Module 2: Flow Classifier. This module classifies individual packets into flows. The classifier accepts a packet and, using a hash function, produces a unique 16-bit flow identification for the packet based on a five tuple value. In our implementation, N_{max} amounts to F_{data} which is 65,536, restricted to this only by the available block RAM on the FPGA device. We adopt method (2) in Section 3. Therefore, from Equation (2), we determine that the $C_{flow-cycles}$ required is 10. This value rises as the processor engine becomes more complex.

Module 3: Flow Monitor. In this module, flow tracking is performed in parallel. Our implementation uses 10 monitors running concurrently. Although the system is limited to processing 65,536 flows simultaneously, this is a result of the block RAMs

Table 2: Number of cycles to monitor 100 flows, by varying the packets per flow. Total row in the table equates to the number of cycles needed for processing the flows.

Max Flows Packets/Flow	65,536 60	65,536 80	65,536 100
Transfer cycles	7,864,320	10,485,760	13,107,200
Classify cycles	58,982,400	78,643,200	98,304,000
Flow cycles	600	800	1,000
Constant cycles	131,100	131,100	131,100
Total	66,978,420	89,260,860	111,543,300
Speed (50MHz)	2.93 Mpps	2.94 Mpps	2.94 Mpps

available on the FPGA device. The F_{max} obtained after place-and-route from Xilinx tools, is approximately 50 MHz for Virtex XC2V8000. If we assume a P_{ave} of 100 packets per flow, from our implementation, $C_{N_{max}}$ from Equation (3), is approximately 111.5 million cycles. Starting with Equation (4) and given $N_{max} = 65536$, we calculate the overall flow rate $R_{flow-rate}$ for the system is approximately 29,360 flows per second, or 2.94 million packets per second (Table 2).

Module 4: Stats Analysis. Here we simply collate the statistics of each flow and send them to the software module for further analysis.

6 Performance Results

The software element of the system accepts a tcpdump file as input, and then buffers the packets. The packets are then transmitted to the external memory banks as explained in Section 5.

Two parameters affect the results: (i) the transfer rate of packets from large memory, and (ii) the number of packets per flow.

From Equations (3) and (4), we calculate in Table 2 the flow monitoring rates $R_{flow-rate}$ by varying the P_{ave} packets in each flow. The calculations are based upon an F_{max} of 50 MHz for the design, as reported by Xilinx place-and-route tools. It can be seen that our hardware-assisted approach can support up to 3 Mpps, although it is limited to 65,536 active flows, whilst the corresponding pure software prototype only supports 20 Kpps (Section 2); this gives a speedup factor of 150.

Table 2 shows performance of the system based on

Table 3: Effects of (a) removing current FPGA device constraints and (b) optimisation of Modules. Constraints: the limitation of resources available to implement multiple flow modules and store flow statistics. Optimisation: data transfer time between the PC and the Classifier module and the Classifier itself.

	With Constraints	Without Constraints	Optimised Modules
Max Flows Packets/Flow	65,536 100	100,000 100	100,000 100
Transfer cycles	13,107,200	20,000,000	10,000,000
Classify cycles	98,304,000	150,000,000	10,000,000
Flow cycles	1,000	1,000	1,000
Stats cycles	131,100	200,000	200,000
Total	111,543,300	170,201,000	20,201,000
Speed (50MHz)	2.94 Mpps	2.94 Mpps	24.75 Mpps

our analytical model. Here we give further details of how the analytical model can be used to estimate scalability properties of the system if the device constraints are eliminated, and hence provide approximate traffic rates which the system will be capable of handling. There are two main optimisations that can be applied to the system. First, the time taken to transfer the packets from the PC to large memory to the classifier can be eliminated by using an embedded processor, or by constructing dedicated logic for fast transfer of data from external memory to the FPGA logic, or by using a hardware platform with in-built Ethernet access. Second, we envisage the optimisation of the Classifier module, by adopting multiple parallel instances and by undertaking rigorous pipelining of the modules.

If we are able to eliminate the memory resource constraint and to optimise individual modules of the system, we can estimate the resulting performance enhancements to designs shown in Table 2. We assume an average of 100 packets per flow, and the capability of achieving at least 50 MHz FPGA operating frequency. Table 3, column 2, shows the results of calculations for the flow rate without the resource constraint, which allows us to process a large number of active flows in real time. As shown from the table, we can maintain a throughput of approximately 2.94 Mpps for larger amounts of active flows, with the only criterion being the availability of the required resources.

In column 3 of Table 3, we show the effects of reducing the transfer of packets from PC to external memory to classifier. In addition, as suggested above, if the classifier module is heavily pipelined to support classifying a packet in every clock cycle, then the effects of this on the system are remarkable. In fact, any optimisation that reduces the number of clock cycles used for any module increases the throughput of the system.

Table 4: Resource requirements for monitoring large number of flows simultaneously.

Max Flows Packets/Flow	65,536 100	100,000 100	250,000 100	500,000 100
Block RAM	116	177	442	885
Monitor Slices	2,000	2,000	2,000	2,000

It is clear from Table 3 that it is possible to achieve processing rates in excess of 20 Mpps, if we are not constrained by the resources available on the hardware device, or by an inability to optimise individual modules. This is adequate for most networks nowadays, and sufficient for 10 Gbps traffic rates as shown in Table 1.

Table 4 shows the amount of resources that would be required for large active flows to be processed at the estimated performance rate.

7 Run-time Reconfiguration

The modules described in Section 3 and Figure 3 can be used to provide the flexibility lacking in a full hardware implementation, enabling the system to automatically recognise time-critical occurrences of suspicious packet activity, such as host dropping/losing excessive number of packets, or receipt by a host of unusually large amounts of packets.

The hardware part of the ME may be modified through reconfiguration. We can modify the number of active flows in order to alter the overall size of the ME, or modify the user requirements in order to revise the analysis target(s) and the statistical information stored. There is potential here for tailor-made trade-offs between the speed of the ME and the number of services required to work in parallel. For example, more defined denial of service attack detection or encryption services may be added to the remit of the FPGA if the ME does not use up all the available hardware capacity.

To determine the effects of the proposed reconfigurations on an ME, we can calculate the time taken, $T_{full-config}$, to fully reconfigure the FPGA using the following equation:

$$T_{full-config} = S_{config}/F_{recon} \quad (7)$$

where S_{config} is the total configuration size in bytes of the FPGA and F_{recon} is the reconfiguration frequency in bytes per second.

Such run-time reconfiguration may have overheads: for example, during reconfiguration, it is possible that packets cannot be processed. Two ways of addressing this problem are that we can (a) accept the inevitable loss of some packets during this time, or (b) buffer the packets with the software module of the system. This will be a decision for the end-user, but ideally it is desirable to reconfigure in a time that would allow on-chip buffering. We can derive the required size of buffer memory, S_{buf}, with the following equation:

$$S_{buf} = R_{flow-rate} \times T_{full-config} \times B_{packets} \quad (8)$$

Full reconfiguration is relatively simple; many systems use the Xilinx SelectMap interface for reconfiguration. 8-bits of data can be written every clock cycle, and the typical average clock speed for reconfiguration is 50–60 MHz.

Using Equation (7), we can determine the length of time taken to make modifications to the number of modules, as well as to add another service to the FPGA function. We calculate the reconfiguration time taken, and its impact, by altering the analysis requirements as well as adding a payload pattern matching engine to the FPGA. The total configuration bits for a Virtex II XC2V8000 are approximately 26,194,208 bits [11], and given an average reconfiguration clock speed of 50 MHz, the reconfiguration time, $T_{reconfig}$, is approximately 65.5 ms. We calculate that approximately 192,500 packets could have been processed within that time. In such circumstances, we must either buffer the packets or accept packet loss during reconfiguration. If the packets are buffered, this will require some resources. Using

Equation (8), we calculate that approximately 7.7MB of memory is required to store the packets.

This result is for full reconfiguration; reconfiguration time will be far less if the FPGA supports partial reconfiguration. For partial reconfiguration in a Virtex device, we must decide how many frames we would like to reconfigure, as different devices have different frame sizes - for example the XC2V8000 has 2860 frames and each frame has 9152 bits.

We can express the partial reconfiguration as:

$$T_{part-config} = Q_{columns} \times S_{config}/F_{recon} \quad (9)$$

where S_{config} in this case is determined by:

$$S_{config} = N_{frames} \times L_{length} \quad (10)$$

For the device we used the XC2V8000 has 46,592 slices, in an array of 112 rows by 104 columns [11]. On average, 1 CLB column = 46592/104 = 448 slices, approximately 27.5 frames. So with our design taking approximately 11,000 slices, we need a minimum of approximately (11000/448) = 25 columns. We can derive the required buffer space needed as follows.

$$S_{config} = 28 \times 9152 = 32032bytes \quad (11)$$

So, the configuration time for one column is 32032/50MHz = 0.64ms. Therefore, for 25 columns, the partial configuration time (Equation (9)), is:

$$T_{part-config} = 25 \times 0.64 = 16ms \quad (12)$$

Hence, using Equation (8), S_{buf} requires approximately 1.9MB, which is becoming feasible for storage using memory on the FPGA.

8 Summary

We have endeavoured to describe how statistics gathered from monitoring flow traffic between endpoints on a network can be employed to better utilise bandwidth and resources, and to locate areas of possible security failure and intrusion incidents. Having found that current software-based systems lack the ability to process flows at current link speeds of gigabit rate, we develop a combined hardware-software measurement engine which supports effective flow processing between the endpoints in a networking environment at gigabit rates.

We are exploring the development of the monitoring architecture to provide Quality of Service (QoS) and Denial of Service (DoS) detection. There is also the possibility of detecting new threats and attacks based on the approach introduced in this paper: the amount of hardware and software resources, and the use of run-time reconfiguration, can be adjusted to meet the characteristics of such threats and attacks.

Acknowledgements. The support of UK EPSRC (Grant numbers GR/R 31409, GR/R 55931 and GR/N 66599), EU Diadem Firewall project, Celoxica and Xilinx is gratefully acknowledged. Our thanks also goes to Sockeye Networks for providing the ME software.

References

[1] K.G. Anagnostakis et al, "Open Packet Monitoring on FLAME: Safety, Performance and Applications", *Proc. 4th International Working Conf. on Active Networks (IWAN'02)*, 2002.

[2] P. Bellows et al, "GRIP: A Reconfigurable Architecture for Host-Based Gigabit-Rate Packet Processing", *Proc. IEEE Symp. on Field Programmable Custom Computing Machines*, 2002.

[3] Y. Breitbart et al, "Efficiently Monitoring Bandwidth and Latency in IP Networks", *Proc. IEEE Infocom Conf.*, 2001.

[4] E. Kohler, R. Morris, B. Chen, J. Jannotti and M.F. Kaashoek, "The Click Modular Router", *ACM Transactions on Computer Systems*, volume 18, number 3, 2000, pp. 263–297.

[5] M. Necker et al, "TCP-Stream reassembly and state tracking in hardware", *Proc. IEEE Symp. on Field Programmable Custom Computing Machines*, 2002.

[6] T.H. Ptacek and T.N. Newsham, "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection", Technical Report, Secure Networks, Inc., 1998. citeseer.nj.nec.com/ptacek98insertion.html, 1998

[7] D.V. Schueler and J. Lockwood, "TCP-Splitter: A TCP/IP Flow Monitor in Reconfigurable Hardware", *Proc. 10th Symposium on High Performance Interconnects Hot Interconnects*, 2002.

[8] J. Gause, P.Y.K Cheung, and W. Luk, "Reconfigurable shape-adaptive template matching architecture", *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, IEEE Computer Society Press, 98–107, 2002.

[9] Cisco Systems, NetFlow, http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm.

[10] FlowScan Network Analysis Tool, http://net.doit.wisc.edu/ plonka/FlowScan.

[11] Xilinx, www.xilinx.com.