Reconfigurable Architecture for Network Flow Analysis

S. Yusuf, W. Luk, M. Sloman, N. Dulay, E. C. Lupu, and G. Brown

***Abstract*—This paper describes a reconfigurable architecture based on field-programmable gate-array (FPGA) technology for monitoring and analyzing network traffic at increasingly high network data rates. Our approach maps the performance-critical tasks of packet classification and flow monitoring into reconfigurable hardware, such that multiple flows can be processed in parallel. We explore the scalability of our system, showing that it can support flows at multi-gigabit rate; this is faster than most software-based solutions where acceptable data rates are typically no more than 100 million bits per second.**

***Index Terms*—Flow analysis, flow measurement, network monitor, NetFlow, network security.**

I. Introduction

IT IS NOW common practice to monitor network traffic in order to track statistics on resource-utilization to various destinations by specific protocols. This ability to monitor traffic by endpoints and protocols is a key ingredient in managing bandwidth utilization and costs. To this end, enabling technology such as NetFlow [16] is often used. The NetFlow protocol is implemented by all major router vendors and provides "per flow" packet and byte counts. For the purposes of this paper, a flow is defined as the traffic between a single source and destination (IP address and port) for a single TCP connection.

Although the NetFlow protocol generates byte and packet counts on a per flow basis, the data associated with a flow may not correspond to a complete TCP session between endpoints and, indeed, may include data from multiple TCP sessions. To limit the effects of this, we have developed an experimental measurement engine (ME), which extends the NetFlow technology to provide additional relevant data, such as round trip time (RTT), packet loss, and jitter effects, through the monitoring of TCP packet flows at the protocol level.

The NetFlow protocol, as implemented in a router, generates user datagram protocol (UDP) packets which provide byte and packet counts for traffic between two endpoints. Data collection for NetFlow is generally implemented as a flow cache, in which data corresponding to a flow is maintained as a set of byte and packet counters. NetFlow packets are generated whenever flows are flushed from the cache, for instance, due to inactivity. The individual flows are identified by source and destination addresses and ports, as well as protocol type.

TABLE I
THROUGHPUT WITH DIFFERENT LINK SPEEDS

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

Packet size: 128 bytes. MBPS: million bits per second. KPPS: thousand packets per second. MPPS: million packets per second.

Flow monitoring is important in the detection of suspicious activity which may only become obvious with continuous examination of data. Monitors can automatically alert system administrators after such suspicious activity is detected, and the data collected is subject to analysis by human intervention, with some automated analysis depending on the end user's requirements. More sophisticated systems are even able to isolate affected systems or limit/disable affected services when anomalous activity has been detected.

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 (see Table I). For this reason, most monitors employ packet sampling instead. These systems process thousands of packets per second (Kpps), in contrast to a theoretical rate of millions of packets per second (Mpps) on a 1-Gb/s network, assuming an average packet size of 128 bytes (see Table I).

Conversely, hardware-based systems suffer from limitations in terms of resources. We wish to provide a scalable hardware-based architecture, which can be used to match current and future speed levels of network devices. We present a "real-time" monitoring system based on a combined software/hardware application-specific system which is optimized for both speed and "intelligent" decision-making.

Our approach maps the performance-critical tasks of packet classification and flow monitoring in hardware, such that operations can run in parallel where desirable. We utilize a field-programmable gate array (FPGA) as our hardware target. A key feature of our work is that the architecture relies on being able to process multiple flows in parallel. We generate NetFlow [16] compatible data to provide pertinent flow information, since this protocol is widely implemented by all major router vendors and provides "per flow" packet and byte counts.

Manuscript received May 10, 2006; revised April 22, 2007. This work was supported in part by U.K. Engineering and Physical Sciences Research Council under Grant GR/R 31409, Grant GR/R 55931, and Grant GR/N 66599, by European Framework 6 Project "DIADEM Firewall," by Celoxica, by Synplicity, and by Xilinx.

S. Yusuf, W. Luk, M. Sloman, N. Dulay, and E. C. Lupu with the Department of Computing, Imperial College London, London SW7 2BZ, U.K. (e-mail: sy99@doc.ic.ac.uk; wl@doc.ic.ac.uk; mss@doc.ic.ac.uk; nd@doc.ic.ac.uk; ecl1@doc.ic.ac.uk).

G. Brown is with the Department of Computer Science, Indiana University, Bloomington, IN 47405 USA (e-mail: geobrown@cs.indiana.edu).

Digital Object Identifier 10.1109/TVLSI.2007.912115

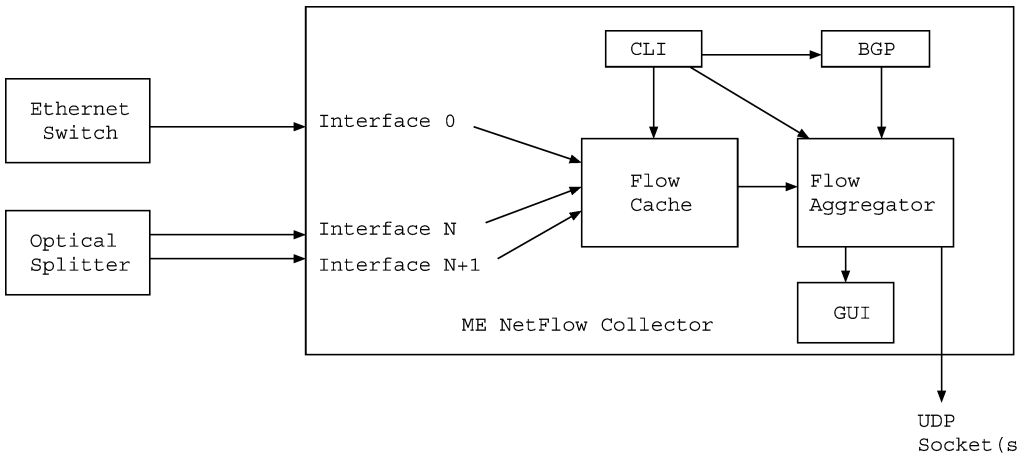


Fig. 1. ME NetFlow System. CLI and GUI, respectively, denote command line interface and graphical user interface. BGP block supports BGP routing.

There has been comparable work carried out on employing reconfigurable hardware as a means of accelerating network processing speed [2]–[4], [8], but none focused on processing of network flows to obtain statistics that could be used to improve network performance.

The following are the three aims of this paper:

1) 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 utilization or packet loss;
2) to address the limitations of software implementations in processing traffic at current network rates;
3) to 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 optimization 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 the following:

1) a network flow analysis architecture based on reconfigurable hardware, which produces flow statistics to identify problematic regions in the network (see Section III);
2) a technology independent model of the system which enables the analysis and prediction of the size and performance of the system (see Section V);
3) a traffic monitoring system which operates on a multi-gigabit network (see Section VII);
4) the use of run-time reconfiguration to provide flexibility for end-user requirements (see Section VIII).

II. 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 utilize the same Internet connection. Thus, all packets must be linked to a particular flow already identified, or used to create a new flow. This process utilizes an inordinately large amount of resources for buffering packet flows, and consumes a great deal of time in determining which flow, if any, a packet belongs to.

Fig. 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 layers 2 and 3 addressing information (MAC addresses, IP address, protocol, and port). The classified flows are then cached and periodically forwarded to the flow aggregator.

The flow aggregator may then optionally provide NetFlow V8 style aggregation by address, port, etc. In addition, the aggregator uses information received through border gateway protocol (BGP) to provide source and destination prefix data. Finally, the output of the aggregator is forwarded through UDP socket(s) to other devices or displayed on a locally generated graphical user interface (GUI). The configuration of the ME NetFlow collector is controlled through a command-line interface (CLI).

The main purpose of the ME is to collect statistics in order to diagnose performance problems such as determining destination regions suffering excessive bandwidth utilization, packet loss, jitter, or long round trip time and also to determine whether the network's security has been breached in some way. This is in contrast to existing NetFlow implementations, which collate certain data in order to provide byte or packet per flow information. Instead, our ME determines the statistics and uses them to identify areas where efficient changes will improve overall security. In addition, as the statistics are being collected, identification of attacks or intrusions will prompt the monitor to alert

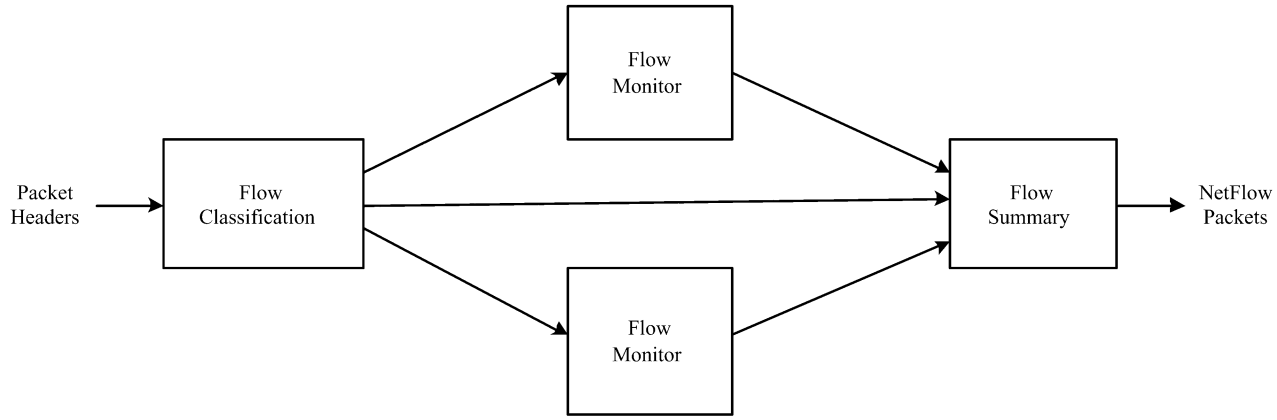


Fig. 2. ME backend.

system administrators in real-time, so that damage to the system can be limited.

As illustrated in Fig. 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. To illustrate the differences in throughput required, consider the processing required to handle 1 Gb/s. This data rate is sufficient to support most enterprises and campuses, although probably not major web-hosting companies. Studies have suggested that this will roughly correspond to 1 000 000 packets/s, but only 10 000 flows/s. Data produced by the University of Wisconsin support this general trend [11]. Undoubtedly, there is no need for the back-end to touch all of the packet data—just the IP and TCP headers—but the processing required for 1 000 000 packets/s is nevertheless a considerable task. Table I shows theoretical throughput for different link speeds.

As the prototype ME is purely software-based, the previously discussed data rates are not attempted. Instead, the prototype is field-tested at a tier 3 ISP where it is capable of handling 10–20 Mb/s of sustained traffic (roughly 20 000 packets/s). The ME prototype back-end is built upon existing software, with packet header capture performed using the Pcap library [13] and packet processing performed with a heavily modified version of TCPtrace [14].

While the prototype ME demonstrates the feasibility of the approach to track packet data and the utility of such data in identifying potential security attacks, in practice, its performance is limited when applied to the Internet. Therefore, software implementations struggle to cope with the typical Internet capacity of 1 Gb/s bandwidth.

For instance, the Click router project involves a highly optimized router in software, which is only able to achieve a rate of hundreds of kpps [3]. However, this requires only packet classification, and not subsequent interpretation of flow data. Although there has been some development in hardware systems capable of packet classification at gigabit rate, these are also limited in that they do not process the classified packets as flows [2], [6], [10].

A view of the flow cache and flow aggregator as ME back-end is illustrated in Fig. 2. Packet headers, consisting of address and port information, flags, etc., 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. For TCP connections, the flow consists of packets in both directions, so that the keys need to be normalized, i.e., the source and destination identifiers must be rearranged in order to identify packets belonging to the same flow.

It may be desirable for the classifier to track all "active flows," where an active flow is defined as a flow still receiving packets. As with NetFlow implementations, a reasonable criterion for active flows is that some packets have been received for the same flow within a period of a few seconds. These active flows are accumulated and buffered until storage capacity runs out. When this occurs, the most outdated flow should be flushed by the cache. Flushing consists of passing the flow header to the appropriate flow monitor for TCP flows, or to the flow-summarizer for other flows. The role of the flow-summarizer is to generate an (extended) NetFlow packet for the flow.

A flow monitor interprets the arriving packets in order to determine relevant statistics. To this end, it must store, for each flow it is assigned, basic TCP header information such as acknowledgements and sequence numbers, as well as all packet headers for which further processing is required (for example, unacknowledged data packets). In general, analysis of TCP behavior can be quite complex [5]. Hence, compromises and inaccuracies must be accommodated if processing is to be handled at the desired rates.

III. Two-Level Flow Analysis Architecture

Our approach consists of an architecture with both hardware and software to support a two-level flow analysis scheme. This combined system allows the processing of flows at rates exceeding 1 Gb/s. It adopts the use of software for less time-critical operations or those which require flexibility, for example, the display of statistics via a GUI, and adopts the use of hardware for those aspects which can be enhanced through parallelism, such as the processing of multiple flows.

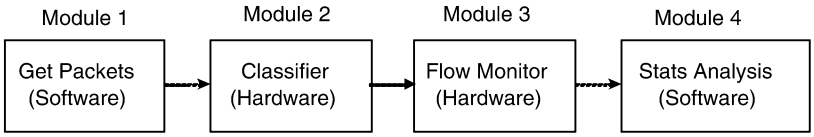


Fig. 3. Hardware and software system overview.

This section describes how our software-based ME can be enhanced to provide the flow monitoring rate for 1 Gb/s traffic. Although a fully hardware-based ME would be quicker, it would also be relatively inflexible, as well as impractical, as there are insufficient resources available on current reconfigurable devices to store sufficient flow information. Our approach, in contrast, involves both hardware and software to provide a cost-effective solution which is also scalable.

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

The **software** element performs further analysis on the statistics derived from hardware. This analysis can be used to determine which areas of the network are suffering from high packet loss or excessive bandwidth utilization. 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 [7]. In addition, inherent software flexibility also allows room for yet-to-be-defined user analysis criteria. We shall explain in Section VIII how run-time reconfigurability of the hardware elements can be exploited to support such user analysis criteria.

We identify and then port to hardware the time-consuming elements of the software implementation (i.e., the classification of packets and the tracking of flows), utilizing fully the parallelism available in hardware, ensuring that we are able to provide processing at high speed, and consequently enabling us to monitor live feeds.

Fig. 3 provides an overview of the system. It is possible for some of the software modules to be replaced by hardware modules described in Sections V and VI. The resulting performance improvement is presented in Section VII.

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. We assume that this delay, between the packet's arrival at the network and when it is transferred to hardware, is negligible in the overall processing of the packets.

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

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 utilization; this constitutes the second-level flow analysis. In Section VIII, we shall explore how this analysis can be used to support recognition of suspicious activities and managed by run-time reconfiguration.

These modules provide the flexibility lacking in hardware and enable our system to automatically recognize time-critical occurrences of suspicious packet activity, such as host dropping/losing excessive numbers of packets, or receipt by a host of unusually large amounts of packets. These activities can be automatically alerted in near real-time to network administrators.

There are several possibilities for implementing the software blocks, such as the following:

- employing the processor in a PC;
- utilizing an embedded processor (for example a Power PC) on an FPGA (for example, the Virtex II Pro);
- 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. In addition, they eliminate the need for a memory transfer as shown in Fig. 4 and allow the transfer of captured packets directly to the classifier, for example, by capturing the packets using an Ethernet module provided by the Xilinx RC300 development board [15].

IV. Hardware Architecture and Analysis

The hardware implementation phase classifies the packets into flows. A flow descriptor is then passed to a flow monitor which tracks the flows and keeps statistics. Our hardware component of the system is therefore comprised of two main modules: the flow classifier module and the flow processor module (see Section VI).

A global view of the architecture of our hardware-based system is illustrated in Fig. 4. The packet header data is read from the memory and used to classify the flows. Once classified, these flows are processed by a flow processing engine, with the resulting data stored in on-chip fast memory. There are $N_{\max}$ flows stored in fast memory; the data in the external memory is used to classify the flows. The value of $N_{\max}$ is determined by the amount of resources available in one or more FPGA devices. Up to $N_{\max}$ flows are processed simultaneously, and statistics from these operations are sent to software.

In this architecture, each flow is allocated to a processing unit, with each flow monitor ultimately multiplexing over multiple flows.

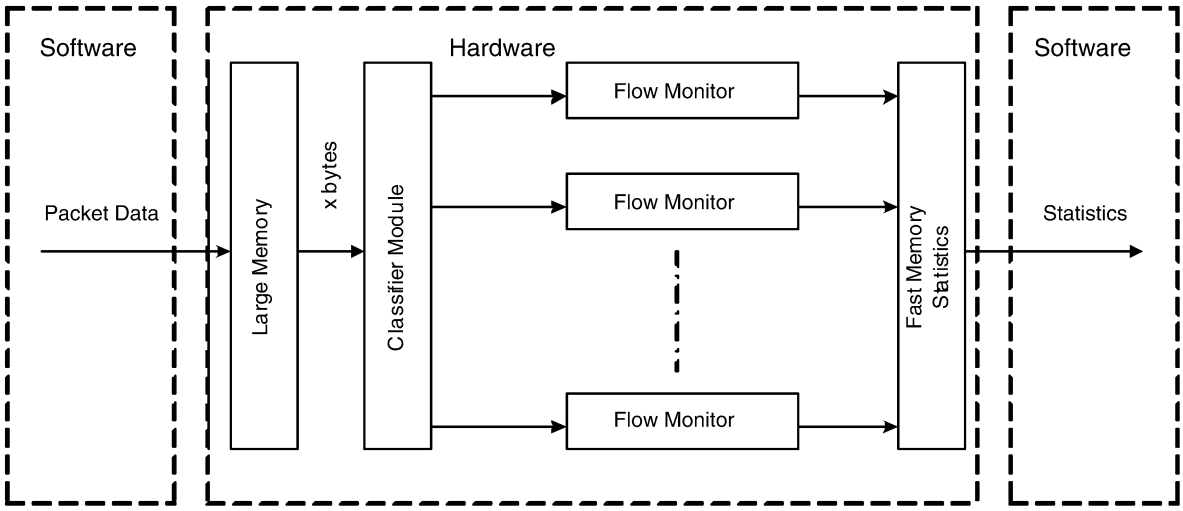


Fig. 4. Architecture of the hardware components of the combined measurement engine.

There are the following 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;
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/s, there may be many times that number of active flows.

In method 1, since flows are buffered before processing, if one wishes to wait for completed flows before processing, then a large amount of storage would be required since many packets are being held for a potentially long time. Method 2 does not suffer from such drawbacks. However, there are inherent difficulties in assigning individual packets to a flow, and then keeping track of the packet data relating to the various flows. Details of the method used for processing are given in Section VI.

V. 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 express the flow processing rate $R_{\text{flow-rate}}$ 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 Fig. 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 $N_{\text{flow-engines}}$ as

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

where L_{total} is the total number of logic cells available on the reconfigurable device. This value of $N_{\text{flow-engines}}$ is used if method 1, stated in Section III, 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

$$N_{\text{flow-engines}} = C_{\text{flow-cycles}} \tag{2}$$

where $C_{\text{flow-cycles}}$ 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, $R_{\text{flow-rate}}$, in terms of the maximum operating frequency $F_{\max}$ of the system

$$C_{N_{\max}} = (P_{\text{ave}} \times N_{\max}) \times (P_{\text{cycles}} + C_{\text{class-cycles}}) + C_{\text{mon-cycles}} + C_{\text{cycles}} \tag{3}$$

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

where $C_{N_{\max}}$ is the total number of cycles to process $N_{\max}$ flows, $C_{\text{class-cycles}}$ is the number of cycles to classify $N_{\max}$ flows based on the average number of packets (P_{ave}), $C_{\text{mon-cycles}}$ 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 $R_{\text{flow-rate}}$, 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, $T_{\text{pkt-trans}}$, 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

$$P_{\text{cycles}} = B_{\text{packets}}/x \tag{5}$$

TABLE II
NUMBER OF CYCLES TO MONITOR 100 FLOWS BY VARYING THE PACKETS PER FLOW

Max Flows	65,536	65,536	65,536
Packets/Flow	60	80	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

Total row in the table equates to the number of cycles needed for processing the flows.

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

$$T_{\text{pkt}-\text{trans}} = F_{\max}/P_{\text{cycles}}. \tag{6}$$

In Section VI, we utilize the equations to show how they are used to estimate the performance of the system, with Table II detailing the results.

VI. 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 six external memory banks. The modules in our architecture, shown in Fig. 4, can be described as follows.

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 (six in total). This uses approximately two clock cycles, P_{cycles} (5), to transfer one packet header, achieving a $T_{\text{pkt}-\text{trans}}$ (6), of up to five million packets per second at an approximate $F_{\max}$ of 10 MHz.

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 (source, destination, source port, destination port, and protocol). The hash function used is a standard RSA hash function. 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 to retain the flow statistics. This figure may vary depending on the width of the hash code and/or the widths of the data for the flows. The data gathered include the number of packets and bytes transmitted per flow, with 16-bit values being used to store this data. We adopt method 2 in Section III. Therefore, from (2), we determine that the $C_{\text{flow}-\text{cycles}}$ required is 10. This value rises as the processor engine becomes more complex.

Each monitor module L_{mon} uses approximately 200 FPGA slices (not including routing resources used) and, from (1), it means the required value of $N_{\text{flow}-\text{engines}}$ is limited to 100. However, one monitor engine takes approximately ten clock cycles, $(C_{\text{flow}-\text{cycles}})$, to process one packet for a flow; this means the required value $N_{\text{flow}-\text{engines}}$ is 10. Since $N_{\text{flow}-\text{engines}}$ needs to be the smaller of the two calculated $N_{\text{flow}-\text{engines}}$, from (1) and (2), therefore, $N_{\text{flow}-\text{engines}}$ in our case is 10. So, in order to process multiple flows simultaneously, the classifier provides multiple flow descriptors to ten flow monitor engines.

Module 3: Flow Monitor. In this module, flow tracking is performed in parallel. Our implementation uses ten monitors running concurrently. Although the system is limited to processing 65 536 flows simultaneously, this is a result of the block RAMs 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 (3), is approximately 111.5 million cycles. Starting with (4) and given $N_{\max} = 65\,536$, we calculate the overall flow rate $R_{\text{flow}-\text{rate}}$ for the system is approximately 29 360 flows/s or 2.94 million packets/s (see Table II).

Module 4: Stats Analysis. Here we simply collate the statistics of each flow and send them to the software module for further analysis. The statistics include the number of packet and bytes transmitted per flow, as well as the packet header data such as the source and destination addresses, ports, etc. The results of the second level analysis will in some cases determine whether or not reconfiguration is required.

VII. PERFORMANCE RESULTS

Our experiment uses a tcpdump file from a server at Imperial College which monitors incoming and outgoing traffic for security reasons. The software element of the system accepts a tcpdump file as input, removes the file header and the tcpdump headers, and then buffers the packets. The packets are then transmitted to the external memory banks as explained in Section VI.

Two parameters affect the results: 1) the transfer rate of packets from large memory and 2) the number of packets per flow. Transferring packets from large memory to the classifier uses a great number of clock cycles, and hence, the number of packets transferred affects the overall flow processing rate. If the only packets to arrive belong to one of the $N_{\max}$ flows, then we would only require sufficient packets to fill each flow. However, not all packets for a flow will arrive consecutively, or even in close proximity to each other. Therefore, it is possible for a packet in a flow that is not already one of the $N_{\max}$ flows to arrive when there is not enough storage room. Nevertheless, to investigate the feasibility of our system, we assume that we would only require sufficient packets to complete $N_{\max}$ flows, as the limits of our hardware resources mean that we would not be able to accommodate any more flows once we have reached our maximum of 65 536.

From (3) and (4), we calculate in Table II the flow monitoring rates $R_{\text{flow}-\text{rate}}$ by varying the P_{ave} packets in each flow, assuming that we only receive packets that belong to one of the $N_{\max}$ flows available. 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, while the corresponding pure software prototype only supports 20 Kpps (see Section II); this gives a speedup factor of 150.

Table II shows performance of the system based on our analytical model. Here we give further details of how the analytical model can be used to estimate scalability properties of the

TABLE III
EFFECTS OF (A) REMOVING CURRENT FPGA DEVICE CONSTRAINTS AND (B) OPTIMIZATION OF MODULES

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

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.

system if the device constraints are eliminated, and hence, provide approximate traffic rates which the system will be capable of handling. The main constraint which affects the system is the area resource available to implement multiple flow modules and the amount of flow statistics that can be stored in the fast memory. In addition, the modular architecture of our system allows for the optimization of individual modules.

There are the following two main optimizations that can be applied to the system:

1) 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;
2) we envisage the optimization of the Classifier module, by adopting multiple parallel instances and by undertaking rigorous pipelining of the modules. In other words, improvements can be made with the availability of a sufficient amount of fast memory, as well as by optimizing one or more modules in the system.

To illustrate the effects of eliminating the memory resource constraint together with further optimizations of individual modules, we provide below an estimate of performance enhancements to the designs shown in Table II. We assume an average of 100 packets per flow and the capability of achieving at least 50 MHz FPGA operating frequency. Table III, 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 Table III, 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 III, we show the potential transfer rate of using a hardware platform with access to the Ethernet, which could result in transfer of a packet per clock cycle from PC to external memory to classifier. In addition, as suggested before, 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 optimization that reduces the number of clock cycles used for any module increases the throughput of the system.

TABLE IV
RESOURCE REQUIREMENTS FOR MONITORING LARGE NUMBERS 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 III 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 optimize individual modules. This is adequate for most networks nowadays, and sufficient for 10 Gb/s traffic rates as shown in Table I.

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

These results indicate that we are capable of using our architecture to determine the hardware requirements for different throughput, or to determine the throughput achievable based on available resources and module capabilities.

VIII. RUN-TIME RECONFIGURATION

The modules described in Section III and Fig. 3 can be used to provide the flexibility lacking in a full hardware implementation, enabling the system to automatically recognize 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. These activities can be automatically notified in near real-time to network administrators [7].

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 tradeoffs 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, $T_{\text{full-config}}$, to fully reconfigure the FPGA using the following equation:

$$T_{\text{full-config}} = S_{\text{config}}/F_{\text{recon}} \tag{7}$$

where S_{config} is the total configuration size in bytes of the FPGA and F_{recon} is the reconfiguration frequency in bytes per second. For example, if we choose to partially reconfigure the FPGA, if only minor modifications are required, then it takes less time, assuming the FPGA supports partial reconfiguration.

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 1) accept the inevitable loss of some packets during this time or 2) 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:

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

If we reconfigure the system, for instance by reducing some of the modules to create space, to include a custom built intrusion detection system to eliminate an attack, then once this is done we can reconfigure again to restore the system back to normal.

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. However, on a Virtex 4 device, for example, the ICAP port can be 32 bits wide [19] and can be clocked up to and in excess of 100 MHz. This reduces the configuration time and the buffer size required.

Using (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 26 194 208 bits [20], and given an average reconfiguration clock speed of 50 MHz, the reconfiguration time T_{reconfig} is 65.5 ms. We calculate that 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, using (8), we calculate that 7.7 MB 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

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

where S_{config} in this case is determined by

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

This is with the assumption that the design is precisely packed onto reconfiguration frame boundaries. For the device, we used the XC2V8000 has 46 592 slices, in an array of 112 rows by 104 columns [20]. On average, 1 CLB column $= 46\,592/104 =$ 448 slices, approximately 27.5 frames. Our design takes approximately 11 000 slices, therefore, a minimum of $(11000/448) =$ 25 columns are required. We can derive the required buffer space needed as follows:

$$S_{\text{config}} = 28 \times 9152 = 32\,032 \text{ bytes}. \quad (11)$$

So, the configuration time for one column is $32\,032/50$ MHz $=$ 0.64 ms. Therefore, for 25 columns, the partial configuration time (13), is

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

Hence, using (8), S_{buf} requires 1.9 MB, which is more feasible for storage using memory on the FPGA. For example, by applying (13) to the use of Virtex 4 XC4VFX140 FPGA

$$T_{\text{part-config}} = 15 \times 18\,902(\text{words})/100 \text{ MHz} = 2.8 \text{ ms}. \quad (13)$$

We note the significant decrease in the time taken for partial reconfiguration [18], [19]. Hence, the S_{buf} required [from (8)] is only 334 KB, which allows the storage, on on-chip memory, of received packets during reconfiguration.

IX. Conclusion

We have described how statistics gathered from monitoring flow traffic between endpoints on a network can be employed to better utilize 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 and software measurement engine which supports effective flow processing between the endpoints in a network at gigabit rates.

Our preferred method makes use of software to capture IP packets, then employs hardware to classify the flows and process them to collate statistics about byte and packet count, potentially also providing statistics about round trip times, retransmissions or packet losses. These statistics are forwarded to software for a second-level analysis, determining areas of high bandwidth utilization or high packet loss or any other user-defined analysis. Our results demonstrate that the proposed hardware-enhanced system has the potential for processing flows at gigabit rates.

Current and future work includes validating our performance estimations using the latest reconfigurable hardware platforms, such as those based on Xilinx Virtex-5 FPGAs. It would also be useful to extend the analysis in this paper to cover the design tradeoffs involved in speed, resource usage, and power consumption. Additionally, 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 our approach: 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.

Acknowledgment

The authors would like to thank Sockeye Networks for providing the initial ME software, upon which they based their experimental ME.

References

[1] K. G. Anagnostakis *et al.*, "Open packet monitoring on FLAME: Safety, performance and applications," in *Proc. 4th Int. Working Conf. Active Netw. (IWAN)*, 2002, pp. 120–131.

[2] D. Nguyen, J. Zambreno, and G. Memik, "Flow monitoring in high-speed networks with 2D hash tables," *Field Program. Logic Appl.*, vol. 3203, pp. 1093–1097, 2004.

[3] E. Kohler *et al.*, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, 2000.

[4] M. Necker *et al.*, "TCP-stream reassembly and state tracking in hardware," in *Proc. IEEE Field Program. Custom Comput. Mach.*, 2002, pp. 286–287.

[5] V. Paxson, "Automated packet trace analysis of TCP implementations," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Arch., Protocols Comput. Commun.*, 1997, pp. 167–179.
[6] G. Memik, S. O. Memik, and W. H. Mangione-Smith, "Design & analysis of a layer seven network processor accelerator using reconfigurable logic," in *Proc. IEEE Field-Program. Custom Comput. Mach.*, 2002, pp. 131–140.
[7] T. H. Ptacek and T. N. Newsham, "Insertion, evasion, and denial of service: Eluding network intrusion detection," Secure Networks, Inc., Calgary, AB, Canada, 1998.
[8] D. V. Schueler and J. Lockwood, "TCP-splitter: A TCP/IP flow monitor in reconfigurable hardware," in *Proc. 10th High Perform. Interconnects Hot Interconnects*, 2002, pp. 54–59.
[9] J. Gause, P. Y. K. Cheung, and W. Luk, "Reconfigurable shape-adaptive template matching architecture," in *Proc. IEEE Field-Program. Custom Comput. Mach.*, 2002, pp. 98–107.
[10] J. van Luteren and T. Engbersen, "Fast and scalable packet classification," in *Proc. IEEE Global Commun. Conf.*, 2003, pp. 560–571.
[11] D. Plonka, "University of Wisconsin network performance statistics," University of Wisconsin, Madison, 2004.
[12] The WinPcap Team, "WinPcap documentation," 2007 [Online]. Available: winpcap.org
[13] LBNL's Network Research Group, Berkeley, CA, "Libpcap," 2007.
[14] S. Ostermann, "TCPtrace," Ohio University, Athens, 2003.
[15] Celoxica Limited, Abingdon, U.K., "RC300: High performance development and evaluation board," 2003, pp. 1–2.
[16] Cisco Systems, San Jose, CA, "Cisco IOS NetFlow—A technical overview," 2006, pp. 1–16.
[17] D. Plonka, "FlowScan: A network traffic flow reporting and visualization tool," in *Proc. 14th Syst. Admin. Conf.*, 2000, pp. 305–317.
[18] Xilinx, San Jose, CA, "Virtex-4 family overview," Tech. Doc. DS112 (v2.0) Preliminary Product Specification, 2007, pp. 1–8.
[19] Xilinx, San Jose, CA, "Virtex-4 configuration guide," Tech. Doc. UG071 (v1.5), 2007, pp. 1–116.
[20] Xilinx, San Jose, CA, "Virtex II platform FPGAs: Complete data sheet," Tech. Doc. DS031 (v3.4) Product Sepecification, 2005, pp. 1–318.

S. Yusuf received the B.Eng. degree in electronic engineering from the University of Leeds, Leeds, U.K., and the M.Sc. degree in computer science and the Ph.D. degree in custom computing for network security applications from Imperial College London, London, U.K.

He was a Research Associate with the Department of Computing, Imperial College London. His current research interests include the areas of network security architecture, telecommunications, information technology, and wireless technology.

W. Luk received the M.A., M.Sc., and D.Phil. degrees in engineering and computer science from the University of Oxford, Oxford, U.K.

He is a Professor with the Department of Computer Engineering, Imperial College London, London, U.K., and a Visiting Professor with Stanford University, Stanford, CA, and Queen's University Belfast, Belfast, Northern Ireland. His research interests include theory and practice of customizing hardware and software for specific application domains, such as graphics and image processing, multimedia, and communications. Much of his current work involves high-level compilation techniques and tools for parallel computers and embedded systems, particularly those containing reconfigurable devices such as field-programmable gate arrays.

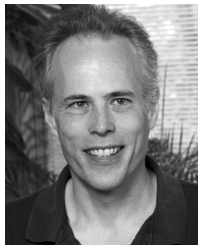
M. Sloman chairs the UKCRC Ubiquitous Computing Grand Challenge steering committee, is a member of the editorial board of the *Journal of Network and Systems Management*, the Advisory Council for INESC-Lisboa, and U.K. Defence Scientific Advisory Council—Information Superiority Board. He was program co-chair for Mobile Data Management (MDM) conference in 2003 and is on the steering committees for the conferences on Policies for Distributed Systems and Networks, Integrated Management (IM), Network Operations and Management (NOMS). His research interests include autonomic management of ubiquitous and distributed systems, adaptive security management, trust and security for pervasive systems. See http://www.doc.ic.ac.uk/~mss for more details and selected papers.

N. Dulay received the B.Sc. degree in computer science from the University of Manchester, Manchester, U.K., and the Ph.D. degree in computing from Imperial College London, London, U.K.

He is a Senior Lecturer with the Department of Computing, Imperial College London. His current research interests include the areas of languages, architectures, and protocols for distributed, mobile, and pervasive systems, particularly to address security, privacy, trust, and context-awareness.

E. C. Lupu received the Diplome d'Ingenieur from the ENSIMAG, Grenoble, France, and the Ph.D. degree from Imperial College London, London, U.K., in 1998.

He is currently a Senior Lecturer with the Department of Computing, Imperial College London, where he leads several research projects in the areas of policy-based network and systems management, pervasive computing, and trust and security funded by the U.K. EPSRC, European Union, and Industry. He has over 60 publications in these areas and serves on the program committee of numerous international conferences including the IFIP/IEEE Symposia on Network Operations and Management, the IEEE International Conference on Self-Adaptive and Self-Organizing Systems and the IEEE Workshop on Policies for Distributed Systems and Networks.



G. Brown received the B.S. degree in engineering from Swarthmore College, Swarthmore, PA, the M.S.E.E. degree from Stanford University, Stanford, CA, and the Ph.D. degree from the University of Texas at Austin, Austin, in 1987.

He is a Professor with the Department of Computer Science, Indiana University, Bloomington. He taught at Cornell University, Ithaca, NY, from 1987 to 1997 and worked as a Research Scientist with Hewlett Packard Laboratories, Cambridge, MA, and an Architect at several networking startups. His research interests include verification and design of digital systems and the preservation of digital documents.