Characterizing Network Processing Delay

Ramaswamy Ramaswamy, Ning Weng and Tilman Wolf
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003
{rramaswa,nweng,wolf}@ecs.umass.edu

Abstract— **Computer networks have progressed from a simple store-and-forward medium to a complex communication infrastructure. Routers in the network need to implement a variety of functions ranging from simple packet classification for forwarding and firewalling to complex payload modifications for encryption and content adaptation. As these functions increase in number and complexity, more processing time is required, and packets experience a significant processing delay. In most network simulations, this delay has not been addressed because it was considered negligible. However, we show that this network processing delay can reach the magnitude of long-distance propagation delay and thus becomes a significant contributor to the overall packet delay. We evaluate different network applications and develop a model that characterizes packet processing cost with only a few parameters that can easily be derived from our simulations. To validate our simulation and our model, we compare them to actual network measurements. The contributions of this work can be used to increase the accuracy of network simulations and improve network performance estimations.**

I. Introduction

The Internet has progressed from a simple store-and-forward network to a more complex communication infrastructure. In order to meet demands on security, flexibility, and performance, network traffic not only needs to be forwarded, but also processed *inside* the network. This packet processing occurs on routers (mainly edge devices) and not on the end-systems. Examples of protocols and applications that require such additional processing are network address translation (NAT), firewalling, and virtual private network (VPN) tunneling. There is also a trend towards more complex services, that demand even more intense processing, like virus scanning, content adaptation for wireless clients, or ad insertion in web page requests. In particular, the need for security in today's Internet is leading towards more processing on edge and access routers where traffic can be filtered and blocked if necessary. This trend towards increasing computation will continue as more security features and services will have to be implemented in the future. The packet delay caused by this processing on routers is the topic of this paper.

To handle the increasing functional and performance requirements, router designs have moved away from hard-wired ASIC forwarding engines. Instead, software-programmable "network processors" (NPs) have been developed in recent years. These NPs are typically single-chip multiprocessors with high-performance I/O components. A network processor is usually located on each input port of a router. Packet processing tasks are performed on the network processor before the packets are passed through the router switching fabric and on to the next network link. This is illustrated in Figure 1.

Due to the increasing complexity of processing, routers require more time to forward packets. The required performance is achieved by processing many packets in parallel on the set of processor cores. This improves the overall router throughput and supports increasing link speeds. Individual packets, however, observe increasing delays because they are processed on a single processing core. Together with the increasing complexity of packet handling, this causes the processing delay on networks nodes to become increasingly important. In this paper, we characterize this delay in more detail.

To illustrate the impact of processing delay or "processing cost," we briefly discuss the various network delays that contribute to the overall packet delay. When sending a packet from one node to another, the following delays occur: (1) transmission delay (the time it takes to send the packet onto the wire), (2) propagation delay (the time it takes to transmit the packet via the wire), (3) processing delay (the time it takes to handle the packet on the network system), and (4) queuing delay (the time the packet is buffered before it can be sent). Table I shows a simple back-of-the-envelope calculation for these delay components for a 1Gbps link, a 1250 byte packet and a 200km link. In most cases, the key contributors of delay are (2) and (4) and are therefore considered in simulations and measurements. The transmission delay (1) is usually small for fast links and small packets and is therefore not considered. Traditionally, the processing delay (3) has also been negligible (as shown in column "Simple Packet Forwarding"). We show, however, that this is not the case anymore as packet processing on routers becomes more complex. Our measurements and simulations indicate that packet processing can take considerable time when payload modifications are involved. Encryption of a single packet, for example, can take in the order of milliseconds, which contributes as much as 50% of the overall packet delay (as shown in column "Complex Payload Modifications" in Table I).

First, we discuss related work in Section II. We then discuss how to estimate the processing delay of packets in Section III using our tool called PacketBench. We develop an analytic model to estimate processing delay for different network applications in Section IV. To verify the accuracy of this approach, we compare our results to measurements of processing delays

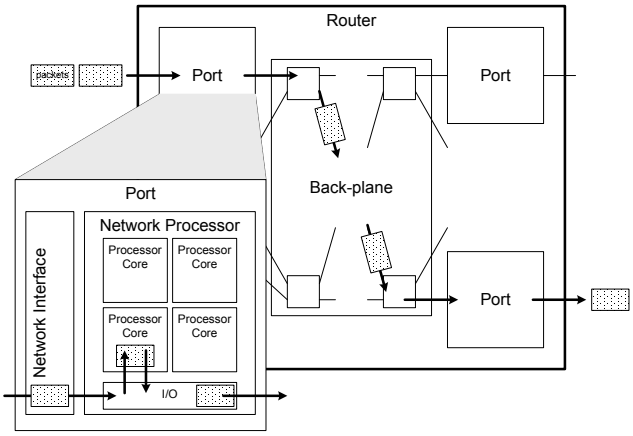


Fig. 1. Packet Data Path on Network Router. Packets are shown as shaded boxes. Packet processing is performed on a network processor that is located at the input port of the system.

Delay	Simple Packet Forwarding	Complex Payload Modifications
Transmission delay	10μs	10μs
Propagation delay	1,000μs	1,000μs
Processing delay	10μs	1,000μs
Queuing delay	$0 \ldots \infty$	$0 \ldots \infty$
Fraction of processing delay to total delay	$\sim$1%	$\sim$50%

TABLE I

NETWORKING DELAY COMPONENTS. A 1GB/S LINK, 1250 BYTE PACKET, 100MIPS PROCESSOR, AND LINK DISTANCE OF 200KM ARE ASSUMED.

on an actual router in Section V. Finally, we show that the use of this model can improve network simulations to accurately reflect the impact of processing delay. Section VI summarizes and concludes this paper.

II. RELATED WORK

There are numerous examples of packet processing on network nodes that extend the basic packet forwarding paradigm. Routers can perform firewalling [1], network address translation (NAT) [2], web switching [3], IP traceback [4], and many other functions. In the context of security, routers need to handle virtual private networks (VPN), where packets are encrypted and tunneled over the Internet to achieve secure communication between trusted subnets and hosts. The processing required for VPN termination is significant as it requires the use of cryptographic algorithms (e.g., Advanced Encryption Standard (AES) [5]).

In recent years, network processors have become available for performing general-purpose processing on high-bandwidth data links. NPs are system-on-a-chip multiprocessors that are optimized for high-bandwidth I/O and highly parallel processing of packets. A few examples are the Intel IXP1200 [6] and EZchip NP-1 [7]. While NPs are the most realistic network processing platforms, one of their main challenges is that they require a significant amount of hardware expertise to be programmed due to their complex and heterogeneous system architecture. Therefore, we cannot easily use NP simulators to derive quantitative results for developing network processing delay models. Instead, we have developed our own tool called PacketBench (discussed in Section III), which is much simpler to use but still yields suitable simulation results.

In the context of network processors, processing complexity has been discussed in several benchmarks. Crowley *et al.* has defined simple programmable network interface workloads in [8]. Wolf *et al.* have developed a network processor benchmark called CommBench [9]. Memik *et al.* have proposed a similar benchmark more recently [10]. All these benchmarks are useful in that they define a realistic set of network processing applications, but are limited as their results do not directly translate into a networking related metric (e.g., packet delay). The processing delay model that we develop in Section IV addresses this issue. Recently, Choi *et al.* [11] have proposed a delay model similar to ours for backbone routers. They study the variation in total point-to-point delay in a backbone network, while we focus on the delay encountered by a packet when it is processed in different ways by a router.

It is particularly important to consider processing delay in network simulations. However, the capability to specify processing cost in current network simulators is limited. The *ns-2* [12] network simulator does not consider processing delays, but can be extended to do so. The NEST network estimator [13] provides a *slumber()* method which can be used to suspend node execution for a definite period of time. The OPNET network simulator [14] can simulate processes which are behavioral descriptions of the functionality of network nodes. What is still missing, is a simple model for processing delay that can be easily integrated into these simulators. Our model is very suitable for this purpose as it uses only two parameters and can easily be adapted to different systems. To show this, we have extended *ns* to simulate processing delay accordingly.

III. PACKET PROCESSING SIMULATION

We have developed a tool called PacketBench [15] that gives us the ability to accurately measure the processing performed on packets in real network traces. PacketBench is simulated on the ARM [16] target of the SimpleScalar [17] simulator to get statistics such as the number of instructions executed and the number of memory accesses made. This simulator was chosen because the ARM architecture is very similar to the architecture of the core processor and the microengines found in the Intel IXP1200 network processor.

We simulated four different network processing applications using PacketBench, which range from simple forwarding to complex packet payload modifications. The simulations were performed on a set of actual network packet traces from both a LAN and the Internet[18]. The specific applications are:

- **IPv4-radix.** IPv4-radix is an application that performs RFC1812 compliant packet forwarding[19] and uses a radix tree structure to store entries of the routing table. The routing table is accessed to find the interface to which the packet must be sent, depending on its destination IP address.

- **IPv4-trie.** IPv4-trie is similar to IPv4-radix and also performs RFC1812-based packet forwarding. This implementation uses a trie structure with combined level and path compression for the routing table lookup [20].
- **Flow Classification.** Flow classification is a common part of various applications such as firewalling, NAT, and network monitoring. The packets passing through the network processor are classified into flows which are defined by a 5-tuple consisting of the IP source and destination addresses, source and destination port numbers, and transport protocol identifier.
- **IPSec Encryption.** IPSec Encryption is an implementation of the IP Security Protocol [21], where the packet payload is encrypted using the 3DES (Triple-DES)[22] algorithm. This algorithm is used in most commercial broadband VPN routers. This is the only application where the packet payload is read and modified.

Using PacketBench runtime traces, we were able to determine the overall number of instructions that were executed, and the number of memory accesses made in order to process a packet for each of the four applications described above. The metric of *instructions per packet* is a good system-independent measure of processing cost. This metric can also be adapted to various heterogeneous packet forwarding systems. We later show how instructions per packet can be converted to an actual processing time for a particular system.

The variation in processing cost with packet size for the IPv4-radix and IPSec Encryption applications is shown in Figure 2. More detailed results can be found in [15]. From these simulation results, we can make the following observations:

- The number of instructions executed per packet for payload processing applications such as IPSec Encryption is several orders of magnitude higher than the number of instructions required for the other applications which operate only on the packet header. It also increases linearly with packet size.
- In header processing applications such as IPv4-radix and IPv4-trie, the number of instructions executed per packet remains more or less constant, but can show slight variations depending on the destination address of the packet (which may be at different locations in the routing table).
- When comparing the number of memory accesses to the packet size (not shown), trends similar to those in Figure 2 can be observed. In particular, the header processing applications show roughly the same number of memory accesses for all packet sizes. For IPSec, the number of memory accesses exhibits the same linear relationship with increasing packet size as for instruction complexity.

IV. Processing Cost Model

We develop a simple analytic model that describes processing cost as a function of a few parameters and is based on the simulation results from the previous section. First, we develop a simple expression that derives the number of instructions based on packet size. Then we extend this expression to consider the impact of memory accesses as well. Finally, we show how the number of instructions and memory accesses can be translated into an actual processing delay.

Application a	Instructions		Memory	
	α_a	β_a	γ_a	δ_a
IPv4-radix	4,493	0	868	0
IPv4-trie	205	0	50	0
Flow Class.	153	0	79	0
IPSec	-2363	294	-868	104

TABLE II
Application Statistics.

A. Packet Processing Instructions

We use two parameters, α_a and β_a, which are specific to each network processing application a:

- **Per-Packet Processing Cost α_a.** This parameter reflects the instructions that need to be processed for each packet independent of its size (i.e., the y-axis offset in Figure 2).
- **Per-Byte Processing Cost β_a.** This parameter reflects the processing cost that depends on the packet size (i.e., the slope in Figure 2).

The total instructions processed, $i_{a,l}$, by an application a for a packet of length l can then be approximated by

$$i_a(l) = \alpha_a + \beta_a \cdot l. \quad (1)$$

Using PacketBench simulation results for the four applications considered here, we obtain the parameters shown in Table II.

B. Memory Accesses

Similar to the number of instructions, we obtain two parameters for memory accesses for each application a:

- **Per-Packet Memory Accesses γ_a.**
- **Per-Byte Memory Accesses δ_a.**

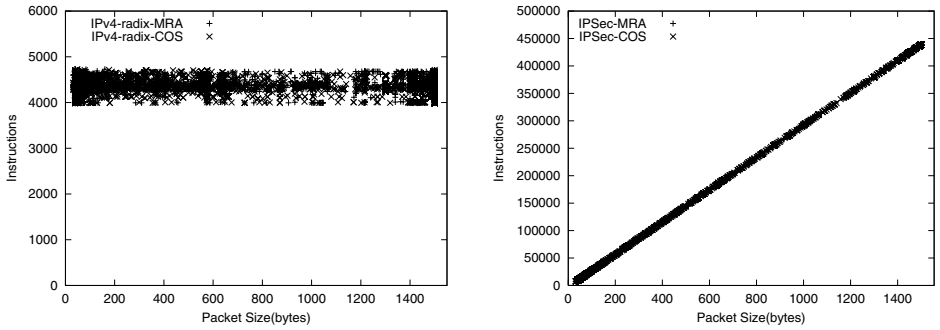
The total number of memory accesses for an application a and a packet of size l is

$$m_a(l) = \gamma_a + \delta_a \cdot l. \quad (2)$$

The memory parameters that we derived from simulation are also shown in Table II. For IPSec, the constant α_{IPSec} and γ_{IPSec} are negative. This is due to the fact that each IPSec packet has a header that is not encrypted and thus does not require cryptographic processing. The packet length, l, includes this header and causes the offset to be negative. Since packets must have a minimum size that includes all headers, the resulting estimate is always positive.

C. System Parameters

In order to derive an overall processing delay as an expression of time, rather than instructions or memory accesses, it is necessary to consider the network system on which the processing is performed. The processing speed of the core and the memory access speed determine the overall processing time.



(a) IPv4-radix (b) Encryption

Fig. 2. Processing Cost over a Range of Packet Sizes. The processing cost is shown for two different trace files.

To capture the processing performance of a RISC core, we use the processor clock frequency, f. This metric is easily obtained and is a good approximation for processing speed. The main problem with using processor clock speeds as performance indicators is that the overall processing time also depends on the processor architecture and other system components. Since basically all network processors today use RISC processor cores, the architecture across systems is very similar. In normal operation, a RISC processor executes one instruction per clock. This yields a processing time, $t_{p,a}$, for application a and a packet of length l of

$$t_{p,a}(l) = \frac{i_a(l)}{f}. \tag{3}$$

However, the processing performance of a RISC processor can be reduced due to pipeline stalls, which occur during memory accesses. There are also other causes for memory stalls, like control hazards, but the impact on the overall performance is less severe than stalls due to memory accesses and are therefore neglected. To integrate the effect of memory delay into our model, we assume an average memory access time of t_{mem} and determine the additional memory access delay, $t_{m,a}$, as

$$t_{m,a}(l) = m_a(l) \cdot t_{mem}. \tag{4}$$

The total packet processing time, t_a, is the sum of both delays:

$$t_a(l) = t_{p,a}(l) + t_{m,a}(l). \tag{5}$$

An example of these system parameters for a network processor (Intel IXP1200 [6]) are: f_{IXP} = 233MHz and t_{mem} = 4ns ... 170ns depending on the type of memory used (on-chip registers vs. off-chip SDRAM). More detailed models for processing performance can be used, but are beyond the scope of this work. Galtier *et al.* have developed a methodology to "translate" performance measurement results from one processor system to another in [23]. Network processor performance models have been discussed in [24].

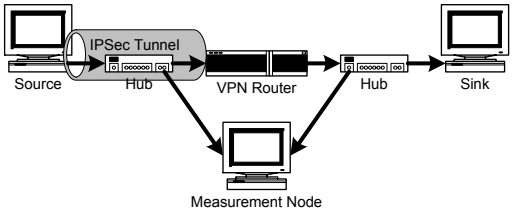


Fig. 3. Measurement Setup. The VPN router is a Linksys BEFVP41 and all network connections are 10Mbps Ethernet.

V. Processing Delay Measurements and Model Verification

In this section, we present more detailed measurements on a network system that show the quantitative impact of processing delay. The results are compared to the simulation and model results from previous sections.

A. Measurement Setup

In this work we are particularly interested in the processing delay on a single router system. In order to isolate and measure the processing delay of individual packets, we need to timestamp packets as they enter and leave the router. We use a commercial off-the-shelf router for this measurement and thus do not have the option of obtaining measurement data from the router. Instead, we use the network setup shown in Figure 3.

Traffic is sent from the source to the sink over 10Mbps Ethernet. The VPN router is a Linksys BEFVP41 system [25]. The measurement node operates both network interfaces in promiscuous mode and can therefore observe the packets that are transmitted on both sides of the VPN router using *tcpdump* [26]. Since the hubs do not buffer the packets, they do not incur any delay between the VPN router and the measurement node.

The delay from the VPN router is measured by timestamping packets on both sides of the router. The difference in the timestamp is the delay. It is crucial that both links are measured by the same computer so that differences in system clocks do not bias the measurement.

The traffic that is sent for the delay measurement is a stream of UDP packets of varying size at a low data rate (a few kbps).

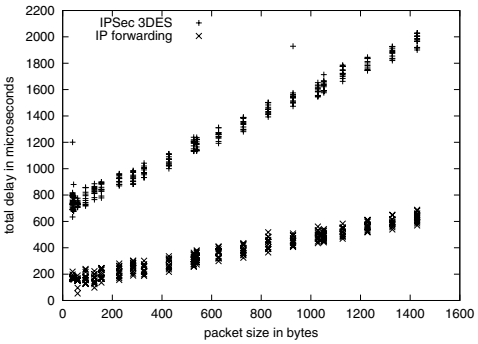


Fig. 4. Measurement Results.

For the round trip time measurement, a TCP connection is used. This keeps queues empty and ensures that we measure the processing delay on the router and not the queuing delay.

B. Measurement Results

We consider two different cases in our measurement:

1) Simple Packet Forwarding. In this case processing is limited to simple IP forwarding. The IPSec tunnel shown in Figure 3 is not used.
2) VPN Termination. In this case all packets require cryptographic processing when moving into and out of the IPSec tunnel.

The results for both applications are shown in Figure 4. The plot shows the processing time in microseconds over the packet size. We can observe that there is an increase in processing time with larger packets – as expected. However, even packet forwarding, which is a simple header processing application, shows this behavior. This is likely due to the fact that the Linksys BEFVP41 system requires memory copies of the packets when moving them from one port to another. This causes larger packets to take additional time, which is not something that is captured in PacketBench. Nevertheless, the overall trend observed in the simulations is clearly visible.

C. Model Use in Network Simulations

The model that we have developed for processing cost can easily be used in network simulations. We show this by adding an estimated delay as derived in Equation 5 to a networking simulation and comparing the results to the measurements we have performed.

We integrated the delay model into the Network Simulator *ns-2* [12] and simulated the topology shown in Figure 3 (excluding the measurement node). The metric that we are interested in is the round-trip time (RTT) for a TCP connection that traverses the VPN router. RTT is a good measure as it directly expresses the end-to-end delay and has a significant impact on the performance of TCP connections.

Figure 5(a) shows the measured RTT for both IPSec and plain forwarding. The RTT for IPSec averages 56.2ms and is about 12ms larger than the 44.1ms RTT for IP forwarding. This is an expected result and confirms the observations from Figure 4 that delay is greater for complex payload processing applications than simple header processing applications.

Figure 5(b) shows the *ns-2* simulation result for the same setup. The x-axis shows time instead of TCP sequence number due to the way *ns-2* reports RTT values. The RTT values are shown as two distinct points due to the resolution at which these values are reported. If a finer resolution was available, the graphs in Figure 5(b) would be similar to those in Figure 5(a). The average RTT for baseline IP forwarding is 42.0ms and thus very close to the measured value of 44.1ms. The RTT increases to 56.9 ms for IPSec. This value is also very close to the RTT observed in the measurement.

These results clearly show that:

- The processing delay on a router has a direct impact on the performance of TCP connections. The processing delay increases the overall RTT and decreases the throughput.
- By extending the network simulator with our simple model for determining processing delay, we can achieve results that capture the measured network behavior.

On the average, a 2-5% error is introduced by using a simple linear model for the instruction counts and memory accesses. We conclude that processing delay needs to be considered in simulations to achieve realistic results and our model can provide good delay estimates for this purpose.

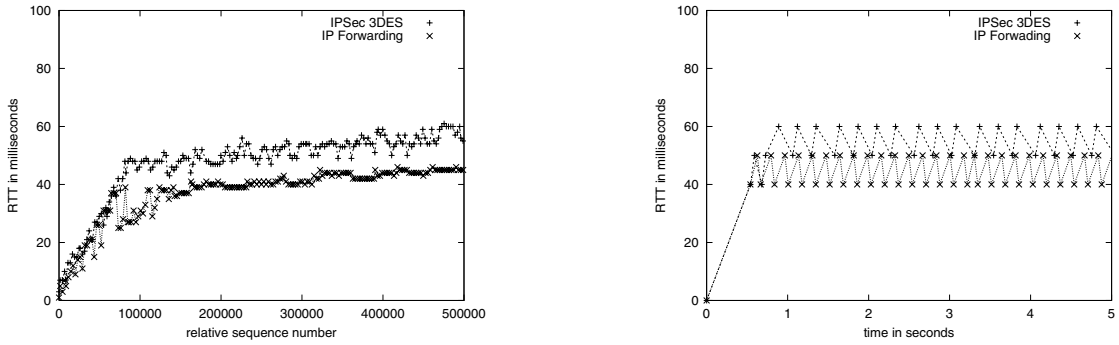
D. Model Complexity

The processing cost model that we have derived is very simple and only requires two parameters. There is a tradeoff between simplicity and accuracy. We feel that it is important to derive a simple model that can easily be integrated into network simulations (as shown above). If the model requires a large number of parameters that are hard to derive and understand, it is unlikely that it will find broad usage. The results from Figures 5(a) and 5(b) also show that even this simple model can improve the accuracy of simulations significantly.

E. Limitations

There are several limitations to this work that we want to point out. First, the performance model in Section IV is only one of many ways of approximating processing cost. Not all applications match the linear behavior that is observed in IP forwarding and IPSec. Examples are flows where processing is unevenly distributed among packets (e.g., HTTP load balancers or web switches [3] where most processing is performed on transmission of the initial URL). However, our model captures two key factors that are characteristic for packet processing: the per-packet delay, and the per-byte delay. Many applications process the headers, which incurs a fixed cost, and some process the payload, which incurs a packet length dependent cost. Therefore we expect that a large number of processing applications fall into the category that can be estimated by our model.

The derivation of system parameters is difficult for network systems where detailed specifications are not available. Also, the use of co-processors and other hardware accelerators leads to heterogeneous architectures that cannot easily be calibrated with a few metrics.



(a) Measurement

(b) Simulation

Fig. 5. RTT for TCP Connection. The simulation, which uses the processing delay model, matches closely the measured network behavior.

Finally, so far we have only explored the impact of processing delay on a small network with a few hops. The next step in this research is to measure the impact of processing on traffic in the Internet and verify the suitability of our model.

VI. Summary and Conclusions

In this paper, we have presented a characterization of network processing delay. This delay is becoming increasingly significant as networks implement more and more complex protocol processing on routers. In order to get a detailed understanding of the characteristics of this processing, we have developed a tool called PacketBench and use simulation results in our analytic processing cost model to derive a simple estimate of processing delay as a function of the packet length and two application parameters. We compare the accuracy of this model with measurements on a real router system. The trends observed in the measurements match those of the simulation indicating that the model is a good approximation. Further, we extended the *ns-2* simulator to include the processing delay model that we have developed. We show that the TCP behavior in the simulation matches the TCP behavior observed in the testbed and significantly improves the accuracy of the simulated results. This shows that it is important to consider network processing delay and that our proposed model is a good approach to achieving a more realistic representation of network processing delay.

References

[1] J. C. Mogul, "Simple and flexible datagram access controls for UNIX-based gateways," in *USENIX Conference Proceedings*, Baltimore, MD, June 1989, pp. 203–221.

[2] K. B. Egevang and P. Francis, "The IP network address translator (NAT)," Network Working Group, RFC 1631, May 1994.

[3] G. Apostolopoulos, D. Aubespin, V. Peris, P. Pradhan, and D. Saha, "Design, implementation and performance of a content-based switch," in *Proc. of IEEE INFOCOM 2000*, Tel Aviv, Israel, Mar. 2000, pp. 1117–1126.

[4] A. S. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer, "Hash-based ip traceback," in *Proc. of ACM SIGCOMM 2001*, San Diego, CA, Aug. 2001, pp. 3–14.

[5] *Advanced Encryption Standard (AES)*, National Institute of Standards and Technology, Nov. 2001, fIPS 197.

[6] *Intel IXP1200 Network Processor*, Intel Corp., 2000, http://www.intel.com/design/network/products/npfamily/ixp1200.htm.

[7] *NP-1 10-Gigabit 7-Layer Network Processor*, EZchip Technologies Ltd., Yokneam, Israel, 2002, http://www.ezchip.com/html/pr_np-1.html.

[8] P. Crowley, M. E. Fiuczynski, J.-L. Baer, and B. N. Bershad, "Workloads for programmable network interfaces," in *IEEE Second Annual Workshop on Workload Characterization*, Austin, TX, Oct. 1999.

[9] T. Wolf and M. A. Franklin, "CommBench - a telecommunications benchmark for network processors," in *Proc. of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Austin, TX, Apr. 2000, pp. 154–162.

[10] G. Memik, W. H. Mangione-Smith, and W. Hu, "NetBench: A benchmarking suite for network processors," in *Proc. of International Conference on Computer-Aided Design*, San Jose, CA, Nov. 2001, pp. 39–42.

[11] B. Choi, S. Moon, Z. Zhang, K. Papagianakki, and C. Diot, "Analysis of point-to-point packet delay in an operational network," in *Proc. of IEEE INFOCOM 2004*, Hong Kong, Mar. 2004.

[12] *The Network Simulator - ns-2*, LBNL, Xerox PARC, UCB, and USC/ISI, http://www.isi.edu/nsnam/ns/.

[13] D. F. Bacon, A. Dupuy, J. Schwartz, and Y. Yemini, "NEST: A network simulation and prototyping tool," in *USENIX Conference Proceedings*, Dallas, TX, 1988, pp. 71–77.

[14] *OPNET Modeler*, OPNET Technologies, http://www.opnet.com, 2003.

[15] R. Ramaswamy and T. Wolf, "PacketBench: A tool for workload characterization of network processing," in *Proc. of IEEE 6th Annual Workshop on Workload Characterization (WWC-6)*, Austin, TX, Oct. 2003.

[16] *ARM7 Datasheet*, ARM Ltd., 2003.

[17] D. Burger and T. Austin, "The SimpleScalar tool set version 2.0," *Computer Architecture News*, vol. 25, no. 3, pp. 13–25, June 1997.

[18] *Passive Measurement and Analysis*, National Laboratory for Applied Network Research - Passive Measurement and Analysis, 2003, http://pma.nlanr.net/PMA/.

[19] F. Baker, "Requirements for IP version 4 routers," Network Working Group, RFC 1812, June 1995.

[20] S. Nilsson and G. Karlsson, "IP-address lookup using LC-tries," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1083–1092, June 1999.

[21] S. Kent and R. Atkinson, "Security architecture for the internet protocol," Network Working Group, RFC 2401, Nov. 1998.

[22] *Data Encryption Standard (DES)*, National Institute of Standards and Technology, Oct. 1999, fIPS 46-3.

[23] V. Galtier, K. L. Mills, Y. Carlinet, S. Leigh, and A. Rukhin, "Expressing meaningful processing requirements among heterogeneous nodes in an active network," in *Proc. of the Second International Workshop on Software and Performance*, Ottawa, Canada, Sept. 2000, pp. 20–28.

[24] M. A. Franklin and T. Wolf, "A network processor performance and design model with benchmark parameterization," in *Proc. of Network Processor Workshop in conjunction with Eighth International Symposium on High Performance Computer Architecture (HPCA-8)*, Cambridge, MA, Feb. 2002, pp. 63–74.

[25] *Linksys BEFVP41 Datasheet*, Linksys, 2004, http://www.linksys.com.

[26] S. McCanne and V. Jacobson, "The BSD packet filter: A new architecture for user-level packet capture," in *Proc. of the USENIX Technical Conference*, San Diego, CA, Jan. 1993, pp. 259–270.