Realizing a Network Emulator System with Intel IXP1200 Network Processor

Go Hasegawa, Haruki Tojo and Masayuki Murata

Graduate School of Information Science and Technology, Osaka University
1-3, Machikaneyama, Toyonaka, Osaka 560-8531, Japan

E-mail: {hasegawa,h-toujou,murata}@nal.ics.es.osaka-u.ac.jp

Abstract— **In this paper, we describe the design and implementation issues of a network emulator system using a network processor. We first describe the required functions for the network emulator system, which include buffering algorithms, and network characteristics emulation. We implemented several functions on the software simulator of the Intel IXP1200 network processor called WorkBench, and confirmed that the implemented functions work correctly. We further obtained the estimated performance of the implemented functions, in order to confirm the processing overhead introduced by those functions is limited when compared with a simple packet-forwarding algorithm.**

Keywords— **Network Processor, Intel IXP1200, Network Emulator, Experimental Network, Micro-engine**

I. Introduction

Against the rapid growth of the Internet population and the explosive increase of the network traffic, research on the Internet has made remarkable progress, especially research focused on increasing link bandwidth, dissolving network congestion, the effective use of network resources, and so on. Consequently, many algorithms, mechanisms and architectures have been proposed and evaluated. Some excellent ones have been introduced into actual networks.

To evaluate the performance of the proposed mechanisms, implementation experiments are necessary in addition to mathematical analyses and computer simulations. However, hardware implementation and experiments have many problems, such as low cost-effectiveness and lack of flexibility and re-usability. Therefore, many researchers have utilized software implementation and experimentation on personal computers (PCs). For example, when evaluating packet scheduling algorithms for the router buffer, PC routers such as ALTQ [1] are often used. In these experiments, the researchers construct a small experimental network and evaluate the proposed mechanisms, since it is very difficult to utilize the large-scale networks that are now in actually operation.

Network emulator systems are categorized into three types: software-based system, hardware-based system, and network processor-based system. Software-based systems are usually implemented as kernel modules on computer operating systems (OSs), such as ALTQ and NISTNet [2]. ALTQ provides various packet-buffering algorithms such as Class-Based Queueing (CBQ) [3], Random Early Detection (RED) [4], RED with In and Out (RIO) [5], and Hierarchical Fair Service Curve (HFSC) [6]. NISTNet has mechanisms that emulate periodical/probabilistic packet losses, link delays, and so on. Although these systems have flexibility in adding/changing implemented mechanisms due to their software rewritability, their packet-processing speed is limited by the performance of central processing units (CPUs) and/or network interface cards on the computers.

STORM [7] and MicroNET [8] systems are examples of hardware-based systems. STORM can simulate up to 10 network nodes and 10 wide area networks (WANs), and realize packet losses, propagation delays, link errors in the network and packet buffering algorithms at gateway buffers. These systems can work much faster than software-based systems since they are implemented by hardware. However, they have little extensibility for adding new mechanisms because of their non-universal design architectures.

On the other hand, network processor-based systems can offer advantages of both the software and hardware-based systems. Because network processors have the hardware and instruction set designed specifically for packet processing, we can construct high-speed network emulator systems. Furthermore, they have large flexibility and extensibility because the functions of the network processors are programmable. Therefore, the development task can be accelerated by sharing the program resources of the network processors, which results in a cost-effective and useful network emulator system for the experimental network. More importantly, the developed mechanism of, for example, a packet-scheduling algorithm can be used in the network processor-based routers with no changes.

In our work, we utilized an Intel IXP1200 network processor [9], and constructed a network emulator system by using IXP1200EB, which is an evaluation board of the IXP1200 [10]. In this paper, we first discuss the functions required for the network emulator system. These include packet-buffering algorithms and an emulation mechanism of network characteristics. We then illustrate the implemen-

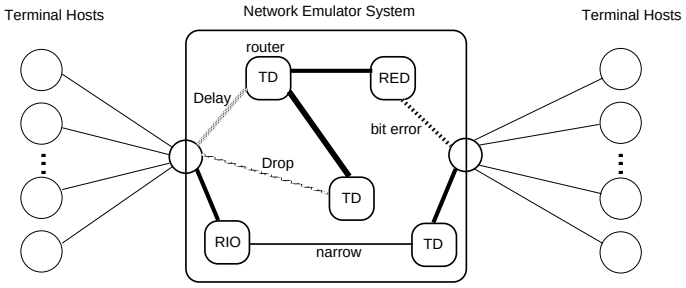

Fig. 1. Network Emulator System

tation designs for these functions on the IXP1200EB. As a basis of our implementation, we used an SRD (Simplified Reference Design) provided by Intel for simple packet forwarding on the IXP1200EB, and we implemented the proposed functions by modifying the SRD implementation. The effectiveness of our system was evaluated by WorkBench [11], which is a software simulator emulating the behavior of IXP1200 perfectly. Through benchmark tests, we confirmed that the processing overhead by introducing proposed functions is limited when compared with the simple packet-forwarding algorithm, which will be reported in Section III.

The rest of this paper is organized as follows. In Section II, we enumerate the functions required for the network emulator system, and illustrate their implementation issues on the IXP1200EB. Section III describes the evaluation results of our system, and discusses the performance of the functions implemented in this work. Finally, we present concluding remarks and discuss future work in Section IV.

II. Network Emulator System

Figure 1 depicts the network emulator system proposed in this paper. The terminal hosts are connected to eight 10/100 base-T Ethernet ports on the IXP1200EB. The network emulator system interconnecting the terminal hosts emulates various network characteristics between the terminal hosts as shown in Figure 1. For example, the propagation delay and/or packet loss probability are set to the link between certain terminal hosts, and a RED router is located as a bottleneck router. The two major network components of our system are packet-buffering algorithms and network characteristics. In this section, we will summarize the packet-buffering algorithms that we implemented. The network emulation functions (such as packet losses and propagation delays) will also be next described. Other features of our network emulator system are also shown in this section.

A. Packet-Buffering Algorithm Emulation

A.1 TD (Tail Drop) Mechanism

In the current Internet, a TD (Tail Drop) mechanism is the most popular as the packet-buffering algorithm at the router. When packets arrive at the router buffer, the TD mechanism simply stores the packets into the buffer in order of arrival, and processes them according to the FIFO (First In First Out) discipline. When the router buffer becomes full because of network congestion, buffer overflow takes place and newly arriving packets are simply discarded. In what follows, we describe the detailed algorithm of the TD mechanism we have implemented on our network emulator system. Note that the network emulator system can arbitrarily set the buffer size for each output port i, which is denoted by B_i.

- Receiver-side algorithm

1. Set the buffer size B_i for each output port i.
2. Prepare the variable q_i, the current queue length (the number of stored packets in the buffer).
3. When a packet arrives at output port j, check the buffer occupancy by comparing q_j with B_j.
4. When $q_j < B_j$, the incoming packet is accepted to the buffer. Increment q_j.
5. When $q_j \geq B_j$, the TD mechanism determines that buffer overflow takes place, and discards the incoming packet. Packet dropping is achieved by releasing memory space assigned to the packet.

- Sender-side algorithm

1. When a packet departs from the output port j, decrement q_j.

A.2 RED (Random Early Detection) Mechanism

RED (Random Early Detection) [4] is now going to be employed to the Internet routers. Before the router buffer becomes full, RED detects the beginning of congestion by monitoring the average queue length at the router buffer and discards incoming packets with a probability determined by a function of the average queue length. More specifically, it uses a low–pass filter with an exponentially-weighted moving average when calculating the average queue length;

$$\overline{q} \longleftarrow (1 - w_q) \cdot \overline{q} + w_q \cdot q \qquad (1)$$

where q is the current queue length at the router buffer, $\overline{q}$ is the average queue length, and w_q is a control parameter. Then RED compares $\overline{q}$ with two thresholds: a minimum threshold (min_{th}) and a maximum threshold (max_{th}) to determine the packet-discarding probability. The detailed algorithm is as follows:

1. When $\overline{q} < min_{th}$, RED stores an incoming packet into the router buffer.
2. When $min_{th} \leq \overline{q} < max_{th}$ an incoming packet is discarded with probability p_a determined by the following equations:

$$p_b \longleftarrow \frac{\overline{q} - min_{th}}{max_{th} - min_{th}} \cdot max_p \quad (2)$$

$$p_a \longleftarrow \frac{p_b}{1 - count \cdot p_b} \quad (3)$$

where max_p is a control parameter to determine the degree of the packet-discarding probability.
3. When $max_{th} \leq \overline{q}$, RED discards the packet.

To implement the above mechanism in our network emulator system, we must treat the real numbers because w_q, $\overline{q}$, p_a, and p_b are all real numbers. However, the micro-engines provided by the IXP1200 can only handle integer numbers. We therefore introduced the fixed point numbers, where the upper 16 bits out of 32 bits are used for representing the integer part of the variables, and the lower 16 bits are used for the decimal part. Thus, our system can handle values ranging from 0.0000152587890625 to 65535.9999847412109375, which is enough to treat control parameters of the RED mechanism. Another problem is that Eqs. (1) – (3) use arbitrary multiplication/division calculations, which takes much more CPU time than addition/subtraction calculations. Therefore, we first convert these equations as follows:

$$\overline{q} \longleftarrow \overline{q} + (q - \overline{q}) \cdot w_q \quad (4)$$

$$p_b \longleftarrow (\overline{q} - min_{th}) \cdot \frac{max_p}{max_{th} - min_{th}} \quad (5)$$

$$p_a \longleftarrow \frac{p_b}{(1 - count \cdot p_b)} \quad (6)$$

Then, by setting the control parameters w_q, max_p, and $(max_{th} - min_{th})$ to the power of 2, and approximating $(1 - count \cdot p_b)$ to the power of 2, these calculations can be complemented only by bit-shift operations. The algorithms of the RED mechanism implemented on our network emulator system are as follows;

- Receiver-side algorithm

1. For each output port i, prepare the control parameters for the RED mechanism ($max_{th,i}$, $min_{th,i}$, $max_{p,i}$ and $w_{q,i}$).
2. For each output port i, prepare the variables $\overline{q}$, $count_i$ and q_i and initialize them to 0.
3. When a packet arrives at output port j, increment $count_j$ and check the average queue length $\overline{q}_j$.
4. When $\overline{q}_j < min_{th,j}$, the packet is accepted to the buffer. Then increment q_j and calculate $\overline{q}_j$ from Eq. (4).
5. When $min_{th,j} \leq \overline{q}_j < max_{th,j}$, the packet-discarding probability is calculated from Eqs. (5) and (6). Then discard the packet with the calculated probability. If the packet is accepted, increment q_j and calculate $\overline{q}_j$ from Eq. (4).
6. When $\overline{q}_j \geq max_{th,j}$, the packet is discarded.
7. When the arriving packet is discarded, $count_j$ is reset to 0.

- Sender-side algorithm

1. When a packet departs from the output port j, decrement q_j.

A.3 Other Mechanisms

Additionally, other kinds of packet-buffering algorithms should be implemented to construct a useful network emulator system; the variants of the RED mechanism such as SRED (Stabilized RED) [12], FRED (Flow RED) [13] and RIO (RED In and Out) for DiffServ architecture, for example.

B. Network Characteristics Emulation

As the network characteristics, the following factors are to be emulated by our system:

Packet Loss for introducing the packet losses caused by network congestion

Propagation Delay for emulating physical distance between hosts

Bit Error for introducing the bit error typically found on wireless/satellite links

Link Bandwidth for emulating a smaller bandwidth link than 10 Mbps, such as a 28.8 Kbps dialup link

Background Traffic Generation for introducing the traffic load to the experimental networks

Packet Filtering for dropping/prioritizing packets from/to specific IP addresses/port numbers

We implemented the emulation of packet loss and propagation delay with fixed control parameters (packet loss probability and delay time). In the following subsections, we explain the detailed algorithms.

B.1 Packet Loss

Packet loss emulation simply discards an incoming packet with a certain probability designated by the system. For the packet loss probability, we can use a fixed value and a value following an arbitrary probability distribution function. We will describe the way to determine the probability distribution function in Subsection II-C.1. Here, we explain the

receiver-side algorithm of packet loss emulation. Note that there is no special procedure required on the sender side.
1. Determine the packet-discarding probability (namely, a fixed value or a probability distribution function) for each output port i.
2. When a packet arrives at the output port, calculate the packet discarding-probability of the packet from the fixed value or the probability distribution function.
3. Determine whether the packet is discarded or not by using the calculated packet discarding probability.
4. When the packet is accepted, store it to the buffer and increment q_i.
5. Otherwise, discard the packet by releasing the memory space where the packet is stored.

B.2 Propagation Delay

In the propagation delay (link delay) emulation, the network emulator system delays the emission of the arriving packets during a certain time to emulate the link distance between the hosts. As in the packet loss emulation, we can use the fixed value or the value following an arbitrary probability-distribution function for the delay time.

For achieving the link delay emulation, we prepare an additional queue called the *delay queue* in the memory space. Packets that require delay are stored in the delay queue with the calculated delay time. Note that the delay time of each packet is represented by the offset from that of the preceding packet. At some intervals, the delay time in the queue is checked from the beginning of the queue the packets are output when it expires. The detailed algorithm is as follows:
1. Determine the delay time (a fixed value or a probability distribution function) for each output port i.
2. When a packet arrives, calculate its delay time from the fixed value or the probability distribution function.
3. The packet is stored to the delay queue with the calculated delay time.
4. For each packet arrival, check whether the delay time has expired or not from the beginning of the delay queue.
5. When the delay time expires, the packet is moved from the delay queue to the output port buffer, and the next packet is checked.

C. Other Features

C.1 Probability Distribution Function

As described in the previous subsections, out network emulator system can use arbitrary probability-distribution functions for determining the packet loss probability and the packet delay time. However, probability-distribution functions such as a Pareto distribution require complex mathematical calculations, which cause a large overhead for the processing of the IXP1200.

Therefore, in the network emulator system proposed in this paper, we prepare *cumulative distribution function (CDF) tables* in the memory space for storing the precalculated values $(x, C(x))$ of the CDF $C(x)$ corresponding to the designated probability distribution function, where $C(x)$ values in the table are evenly spaced from 0 to 1. When using the probability distribution function, the index value y ($0 \leq y \leq 1$) is first obtained from the random variable, and the variable x that satisfies $y = C(x)$ is obtained. By this mechanism, the IXP1200 can treat the various kinds of probability distribution functions without large calculation overhead.

C.2 Parameter Setting Protocol

The implemented functions for the network emulator system described above have many parameters to be set in advance of the actual experiment: the buffer size B_i for the TD mechanism, max_{th}, min_{th}, max_p and w_q for the RED mechanism, the setting of the CDF table, and so on. These values may be set by modifying the source code of the microengines in the IXP1200, but this is difficult for the users of the network emulator system. Therefore, we prepared a simple protocol called NESP (Network Emulator Setting Protocol) for parameter setting, which is performed by injecting the control packets to the IXP1200EB from an external host connected to the IXP1200EB. Due to space limitation, we omit the detailed descriptions of the protocol.

III. Evaluation and Discussions

We have implemented some of the functions described in Section II. As packet buffering algorithms, TD and RED mechanisms have been realized. We have also implemented the packet loss emulation and propagation delay emulation explained in Subsections II-B.1 and II-B.2, respectively. Due to space limitation, we only show the results of the TD and RED mechanisms and the packet loss emulation in this section. Note that we also confirmed the preciseness and small processing overhead of the propagation delay emulation.

We used the software simulator called WorkBench [11] in the implementation experiments in this section. WorkBench is the development environment for programming on microengines, which are the main CPUs in the IXP1200. It can also simulate the detailed behavior of the IXP1200 and is sufficient for evaluating the processing overhead of the program codes. In this section, we show the evaluation results of the above functions implemented by WorkBench.

For the input traffic to IXP1200 in the experiments, the

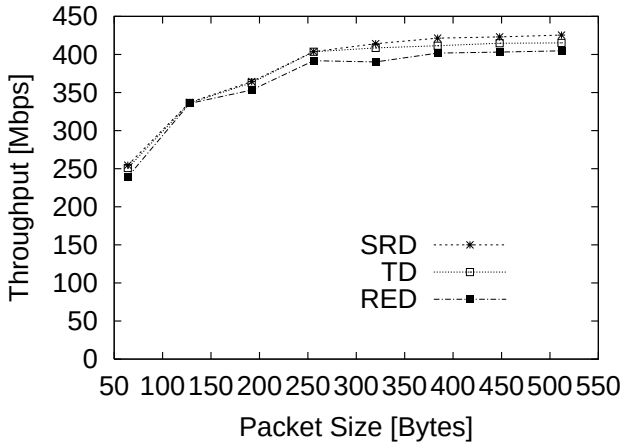

Fig. 2. Packet Size vs. Throughput

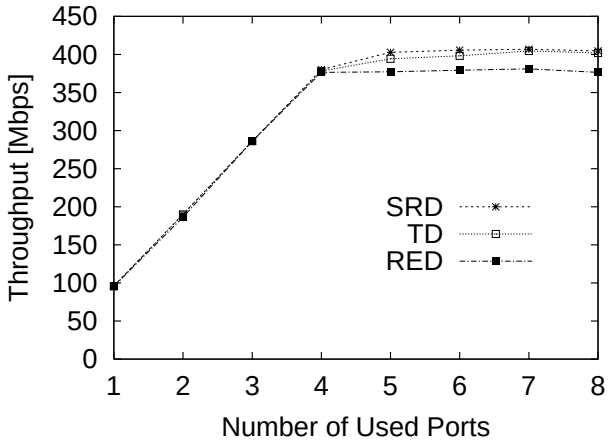

Fig. 3. Number of Input/output Ports vs. Throughput

packet size and inter-arrival time are fixedly set. Although the IXP1200 has six micro-engines, we used only three micro-engines for implementation simplicity. We assigned one micro-engine to the sender-side functions, and two micro-engines to the receiver-side functions. Note that the main purpose of the evaluation in this section is to prove the small processing overhead of our implemented functions, and to confirm that the implemented functions work correctly. We can expect a higher throughput than the following results when all six micro-engines are used, which is beyond of the scope of this paper.

A. Processing Overhead of Packet-Buffering Algorithms

We first evaluated the performance of the TD and RED mechanisms by focusing on the processing overhead of the implementation. For comparison, we also show the experimental results of the SRD, which simply forwards incoming packets without other additional operations.

In Figure 2, we show the throughput values of the SRD, TD, and RED mechanisms as a function of packet size of the input traffic. Here, we used eight ports for packet input/output, and the input rate of each port is fixed to 100 Mbps. Therefore, the total input rate is 800 Mbps. From the results, we observe that all three mechanisms show almost the same throughput values regardless of packet size. This means that our implemented TD and RED mechanisms work well with small processing overhead. We further see that the throughput values of the three mechanisms are all bounded at about 430 Mbps, even when we use large-sized packets. This is considered to be the performance limit of the three mechanisms when we use three out of the six micro-engines in the IXP1200.

We next evaluated the relation between the number of input/output ports and throughput. We fixed the packet size to 256 KBytes, and the input rate of each port is set to 100 Mbps. Figure 3 shows the experimental results. As shown in Figure 2, the three mechanisms have almost the same throughput. This result again confirms the small processing overhead of the TD and RED mechanisms, when compared with the SRD mechanism.

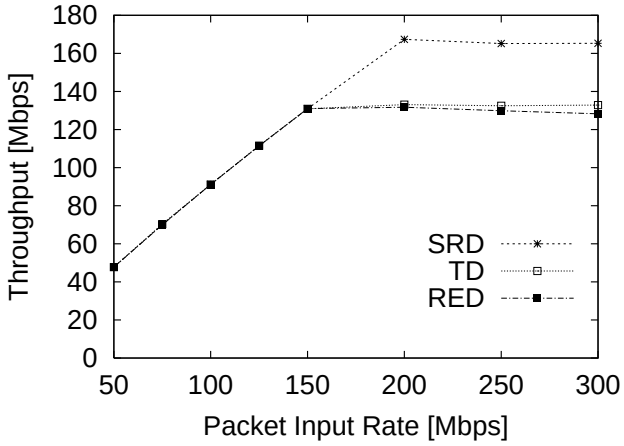

Fig. 4. Input Rate vs. Throughput

We next changed the packet size to 128 KBytes, the number of used ports was fixed to one, and the input rate of the port was changed from 50 Mbps to 300 Mbps. Figure 4 shows the relationship of the input rate and the throughput of the SRD, TD and RED mechanisms. Although they can provide equal throughputs when the input rate is small, the TD and RED mechanisms show slightly smaller throughput values than the SRD when the packet input rate is larger than 150 Mbps. The maximum throughput is about 130 Mbps, which may be the performance limit of packet forwarding

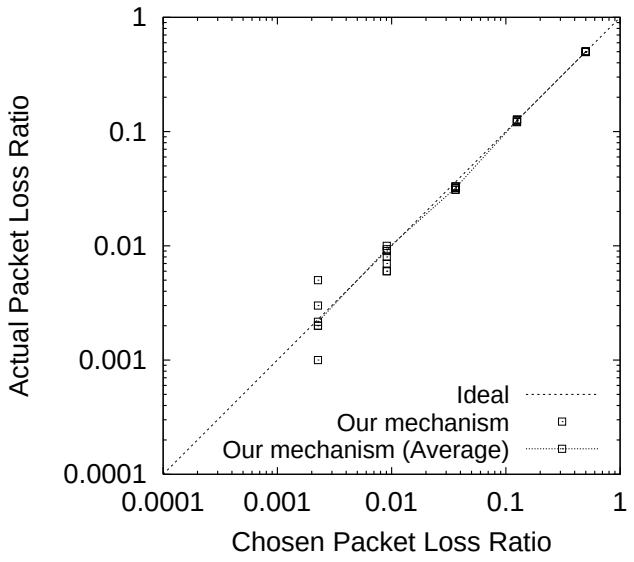

Fig. 5. Preciseness of Packet Loss Emulation

by one port in the TD and RED mechanisms implemented in this work. However, it can fill up the 10/100 Mbps Ethernet port on the IXP1200EB, and we can expect the higher throughput corresponding to the 1000base-SX ports on the IXP1200EB by using all of six micro-engines.

From these results, we conclude that we can provide the TD and RED mechanisms, as packet buffering algorithms, at little expense of the overall packet forwarding performance.

B. Preciseness of Packet Loss Emulation

Lastly, we show the evaluation results of the packet loss emulation explained in Subsection II-B.1. Here we used only one port for packet input/output, and the input rate was set to 100 Mbps. Figure 5 shows the relation between the chosen packet-discarding probability and the actual packet-loss ratio observed from the number of dropped/successfully transmitted packets. From this figure, we can say that the implemented packet loss emulator drops the incoming packets with precise probability. We also confirmed that the packet loss emulation can be implemented with almost the same processing overhead as the TD mechanism, which means that we can introduce packet loss emulation with little performance degradation.

IV. Conclusion

In this paper, we have constructed a network emulator system using the Intel IXP1200 network processor. We have first described the required functions for the network emulator system, and implemented some of them on WorkBench, the software simulator of the IXP1200. We have confirmed that our implemented functions have a small processing overhead, and that they can work without large performance degradation. We have also proposed the additional features for the network emulator system, specifically the way to use probability distribution functions for parameter setting and the protocol for parameter setting of the system from an external host.

We are now implementing the other functions described in this paper. For future work, we also plan to implement our system on the IXP1200EB, and evaluate it through practical experiments.

Acknowledgements

This work was partly supported by the University Program performed by Intel Corporation. We also thank Mr. Akio Tomobe of Intel Corporation for his kind support.

References

[1] ALTQ (Altemate Queueing for BSD UNIX) „ ,” *available from* `http://www.csl.sony.co.jp/~kjc/software.html`.
[2] NISTNet Home Page, ,” available at `http://www.itl.nist.gov/div892/itg/carson/nistnet/`.
[3] Sally Floyd and Van Jacobson, “Link-sharing and resource management models for packet networks,” *IEEE/ACM Transactions on Networking*, vol. 3, no. 4, pp. 365–386, Aug. 1995.
[4] Sally Floyd and Van Jacobson, “Random early detection gateways for congestion avoidance,” *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, Aug. 1993.
[5] D. D. Clark, and W. Fang, “Explicit allocation of best effort packet delivery service,” *IEEE/ACM Transaction on Networking*, vol. 6, no. 4, pp. 362–373, August 1998.
[6] T. S. Eugene Ng Ion Stoica, Hui Zhang, “A hierarchical fair service curve algorithm for link-sharing, real-time and priority services,” in *Proceedings of ACM SIGCOMM ’97*, September 1997.
[7] STORM, ,” available at `http://www.quality-net.co.jp/images/PDF/STORM_web.pdf`.
[8] MicroNET, ,” available at `http://www.adsystems.co.jp/products/micronet/micro_1.html`.
[9] Intel IXP1200 Network Processor Family, ,” available at `http://www.intel.com/design/network/products/npfamily/ixp1200.htm`.
[10] Intel IXP1200 Evaluation Kit, ,” available at `http://www.intel.com/design/network/products/npfamily/eval_kit.htm`.
[11] Intel IXA Software Developers Kit 2.0 for IXP1200, ,” available at `http://www.intel.com/design/network/products/npfamily/sdk2.htm`.
[12] Teunis J. Ott, T. V. Lakshman, and Larry Wong, “SRED: Stabilized RED,” in *Proceedings of IEEE INFOCOM’99*, Mar. 1999.
[13] D. Lin and R. Morris, “Dynamics of random early detection,” *ACM Computer Communication Review*, vol. 27, no. 4, pp. 127–137, Oct. 1997.