

© 2010 Dong Jin

EFFICIENT ETHERNET SWITCH MODELS FOR LARGE-SCALE
NETWORK SIMULATION

BY

DONG JIN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2010

Urbana, Illinois

Adviser:

Professor David M. Nicol

ABSTRACT

Ethernet is the most widely implemented low-level networking technology used today, with Gigabit Ethernet seen as the emerging standard implementation. The backbones of many large scale networks (e.g., data centers, metro-area deployments) are increasingly made up of Gigabit Ethernet as the underlying technology, and Ethernet is seeing increasing use in dynamic and failure-prone settings (e.g., wireless backhaul, developing regions) with high rates of churn. Correspondingly, when using simulation to study such networks and applications that run on them, the switching makes up a significant fraction of the model, and can make up a significant amount of the simulation activity. This work describes a unique testbed that gathers highly accurate measurements of loss and latency through a switch, reports on experiments that reveal the behavior of three commercial switches, and then proposes simulation models that explain the observed data. The models vary in their computational complexity and in their accuracy with respect to frame loss patterns, and latency through the switch. In particular, the simplest model predicts a frame's loss and latency immediately at the time of its arrival, which keeps the computational cost close to one event per frame per switch, provides excellent temporal separation between switches (useful for parallel simulation), and provides excellent accuracy for loss and adequate accuracy for latency.

*To my father, Shunrong Jin
my mother, Peiyun Huang
my fiancée, Xuan Zhuang
for their endless love and support*

ACKNOWLEDGMENTS

First of all I would like to express my sincere gratitude to Professor David Nicol, who has been my adviser since the beginning of my graduate study. He provided me with many helpful suggestions, important advice and constant encouragement during the entire course of this work.

I also wish to express my appreciation to Professor Matthew Caesar, who made many valuable suggestions and gave constructive advice for this work, from testbed setup to data analysis.

Sincere thanks are extended to my colleagues Tim Yardley and David Bergman for numerous stimulating discussions, help with experimental setup and general advice.

Finally, my special appreciation goes to my parents and Xuan for their endless patience, encouragement and love.

This work was supported in part by the National Science Foundation under Grant No. CNS-0524695 and Grant No. CNS-0423431. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

TABLE OF CONTENTS

LIST OF TABLES	vi
LIST OF FIGURES	vii
CHAPTER 1 INTRODUCTION	1
CHAPTER 2 BACKGROUND AND RELATED WORK	4
2.1 Existing Switch Models	4
2.2 Packet Loss Models	5
2.3 Ethernet Models	8
2.4 RINSE Network Simulator	9
CHAPTER 3 MEASUREMENT	11
3.1 Requirement	11
3.2 Testbed	11
CHAPTER 4 ANALYSIS AND MODELING	15
4.1 Data Analysis	15
4.2 Queueing Switch Models	19
4.3 Latency-Approximate Switch Models	23
4.4 Markov Chain Frame Loss Model	30
4.5 Multivariant Gaussian Autocorrelation Model	33
CHAPTER 5 MODEL DESIGN AND IMPLEMENTATION	37
5.1 Switch	37
5.2 Ethernet	39
CHAPTER 6 EVALUATION	48
6.1 Simulation Speed	48
6.2 Frame Loss Modeling Accuracy	50
CHAPTER 7 CONCLUSION AND FUTURE WORK	53
REFERENCES	54

LIST OF TABLES

4.1	NetGear, Output Rates, Three Flows from Three Input Port to One Output Port	18
4.2	3COM, Output Rates, Three Flows from Three Input Port to One Output Port	18
4.3	NetGear, Drop Rate, Three Flows from Three Input Port to One Output Port	19
4.4	Autocorrelation of the Delay Processes $\{X_t\}$ and $\{W_t\}$	33
5.1	Dynamic Table in the Link Object	41
6.1	Performance of Q1, Q2 and Q3 Switch Models for N=20 Topology	49

LIST OF FIGURES

1.1	A Generic Switch Model	2
2.1	Forwarding Device Model in Nohn [2004]	5
2.2	Forwarding Device Model in Roman [2007]	5
2.3	Gilbert Packet Loss Model	7
2.4	K^{th} Order Markov Chain Packet Loss Model, $K=2$	8
2.5	RINSE Network Simulator Architecture	10
3.1	Testbed One Setup	13
3.2	Testbed Two Setup	13
4.1	Frame Delay for Single Flow Traffic	16
4.2	Frame Size vs. Packet Delay for Single Flow Traffic	16
4.3	(a) Output Queue Model for NetGear GS108 (b) Input Queue Model with WRR for 3COM 3CGSU08	20
4.4	Delay/Loss Pattern, Two Flows to One Output Port, (a1) 3COM Real (a2) 3COM Queueing Model (a3) 3COM Simplified Queueing Model (b1) NetGear Real (b2) NetGear Queueing Model (b3) NetGear Analytical Model	22
4.5	Simple Model	24
4.6	Analytical Model Validation	27
4.7	Packet Loss in Time Sequence (0 - Received, 1 - Loss)	30
4.8	State Diagram of Packet Loss Model	31
4.9	Expanded State Diagram of Packet Loss Model	32
4.10	CDF of the Delay Processes $\{X_t\}$ and $\{W_t\}$	35
5.1	Switch Architecture in RINSE	38
5.2	Overview of the Standard Ethernet Model in RINSE	40
5.3	Use Case Diagram for Busy Link	42
5.4	Use Case Diagram for Successful Frame Transmission	43
5.5	Use Case Diagram for Frame Collision	44
5.6	Total Client Throughput - UDP	46
5.7	Wall Clock Time - UDP	46
5.8	Total Client Throughput - TCP	47
5.9	Wall Clock Time - TCP	47

6.1	Architecture for Simulation Speed Tests	48
6.2	3COM 3CGSU08: Performance	50
6.3	Frame Loss Accuracy Evaluation	52

CHAPTER 1

INTRODUCTION

Large-scale networks such as enterprise networks and data centers are frequently built using switched Gigabit Ethernet technology. While the Ethernet standard allows for multiple taps onto a shared line, in switched Ethernet configurations a wired line is dedicated to the connection of the two devices at its endpoints. This essentially eliminates collisions caused by the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) technology in the traditional Ethernet. The behaviors of transport layer protocols (e.g. TCP) and applications (e.g. interactive multi-media applications) are sensitive to loss and delay; it is important to understand these characteristics for developing, testing and validating new techniques and technologies running on large-scale Gigabit Ethernet.

Because of its low cost and flexibility, network simulation is widely used to study protocols and applications running on large-scale networks. However, the cost of simulating the network can easily overwhelm the overall cost of performing the simulation experiment. In a moderately sized network a frame may transit 4 or 5 switches on its journey from source to destination (host to LAN switch, LAN switch to WAN, 2 WAN switches to destination LAN). There can easily be three or four discrete events associated with a frame's passage through the switch (arrival, priority queuing, beginning of transmission, ending of transmission). Twenty events may be involved just to move one frame's worth of information from source to destination. One can build detailed switch models according to the specification in the real products. With high fidelity of architectural specifics comes significant computation cost. However, from an application's point of view traffic might logically be thought of in terms of files or streams. The application events that are really of interest may be a small fraction of the events the simulation is executing. Therefore, we seek an efficient and accurate switch model for simulations where the interest is less on the network and more on the

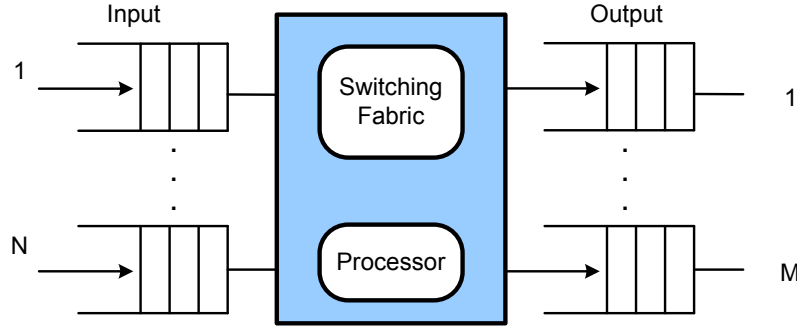


Figure 1.1: A Generic Switch Model

applications and protocols running on the network.

When a frame arrives, a switch examines the destination MAC address in the frame header and uses a forwarding table to determine the output port; if the MAC address is missing, the frame is broadcasted through all other ports. When multiple frames compete for a common output port, switches have different scheduling policies such as first come first serve (FCFS), round robin (RR), weighted fair queueing (WFQ), also known as packet-by-packet generalized processor sharing (PGPS) [1], weighted round robin (WRR) [2], deficit round robin (DRR) [3], virtual clock [4], delay-earliest-due-date (Delay-EDD) [5]. When the buffer is full, any new arrival frames are dropped. The inter-arrival times may be viewed as random—as may processing timing. Therefore, it is natural to use queueing models to describe a switch. Most switches consist of three components: buffers to handle congestion; algorithms to make scheduling and switching decisions; and switching fabric to forward data from one port to another, as shown in Figure 1.1. The buffering strategies include shared buffering, pure input queueing, pure output queueing, input queueing with virtual output queues to avoid head-of-line blocking, and combined input and output queueing. Unfortunately, the precise details of a given switch are often not publicly available, a fact which hinders high fidelity description of a given switch’s operation.

Our goal is to develop a methodology for modeling switches that supports

- fast simulation, e.g., one event per frame per switch,
- accuracy in latency and frame loss prediction that is sufficient for studies of applications and protocols running on the network,

- straightforward model development for new switches.

With respect to accuracy, we prioritize accuracy in frame loss over accuracy in latency. A single frame loss can cause a significant alteration in the size of a TCP send window, whereas the time-scale of activity in an application may be measured in milliseconds while the time-scale of switch latency is a few microseconds. For example, acceptable delay and jitter for audio and video conferencing is on the order of 150 ms and 30 ms respectively according to Cisco’s recommendation [6].

In this work we describe a means of measuring a switch’s latency and frame loss characteristics with extremely high fidelity. We performed comprehensive experiments on commercial gigabit switches to collect traces and obtain sequences of delay and loss pattern. Based on this data, we developed two types of models. One is a simplified queuing model, the other is an algebraic model based on relationship between input rates and output behavior. These models are validated with real traffic, and then compared with each other with respect to execution complexity.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Existing Switch Models

Various types of switch models already exist in network simulators and emulators. Simulators like OPNET [7] and OMNeT++ [8] have detailed switch models that capture complex internal architectures for different types of switches. Significant computation cost is a drawback for this type of models. Also, the detailed specification is often not available to the public. In ns-2 [9] and DETER [10], a simple first-come, first-served (FCFS) queuing model is used for every type of switch. As have others [11], we will see in our experiments for one switch type clear evidence of non-FCFS behavior, such that assumptions of FCFS may produce unrealistic results. Development of device-independent queuing models based on empirical observations was also investigated. Nicolas proposed a simple virtual output queue model as shown in Figure 2.1 [12]. Delay based on empirical data is added to a frame before the frame enters the queue. The model assumes infinite buffer size and thus does not capture frame loss. In addition, the model does not model the interaction among multiple ports. Chertov proposed a queueing model based on experimental data as well [13] (see Figure 2.2). The model places one queue for every port, and fine tunes two metrics: (1) accurate queue size per port and (2) number of servers in the processing unit. Accuracy is improved by taking specific data from real devices into consideration. However, it is still hard to adapt one model to all types of switches once the placement of the queues is fixed. In addition, the simulation speed is slower than that of the simple FCFS queue model.

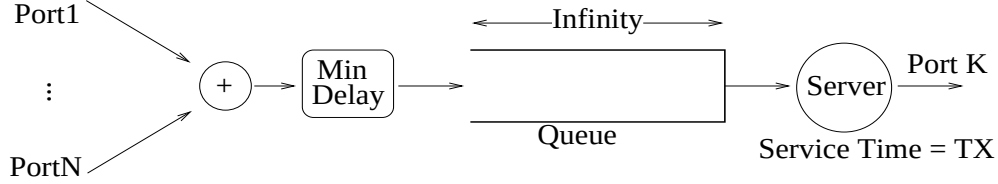


Figure 2.1: Forwarding Device Model in Nohn [2004]

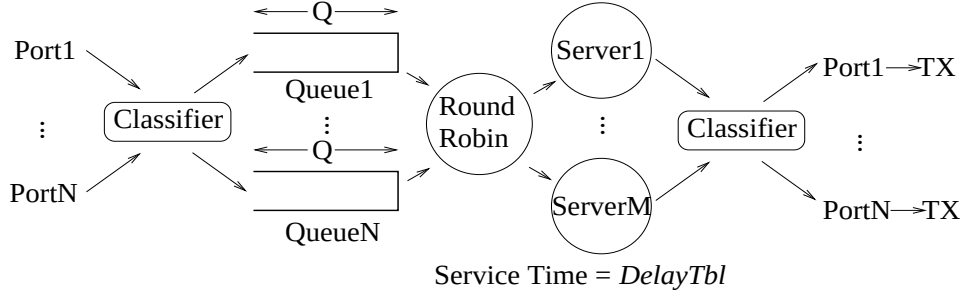


Figure 2.2: Forwarding Device Model in Roman [2007]

2.2 Packet Loss Models

There are two major sources of loss in the computer network: buffer overflow in the intermediate forwarding devices, such as switches and routers, and the lossy links interconnecting various network components.

Buffer overflow is the main cause of packet loss in wireline networks, resulting in more than 99% [14] of all the lost packets. Once the arriving traffic load is higher than the processing and forwarding capacity of the forwarding devices, the packets are queued in the buffers, waiting for their turn to be forwarded on the output link. The size of these buffers is finite. New incoming packets are dropped once the buffer is full. The buffer overflow is a result of limited buffer size and network congestion. Lossy channel or link failures occur more frequently in wireless network rather than wired Ethernet. The high bit-error rates can be caused by noise, interference, and channel fading.

Various packet loss models were proposed. One class is the Markov chain based models, such as Gilbert model [15], extended Gilbert model [16], K^{th} order Markov chain model [17]. Another class uses heavy-tailed distribution

to model the length of consecutive packet loss, such as Pareto distribution [18]. Researchers have also investigated self-similar processes to model traffic in wired line networks [19], [20]. In addition, packet delay and loss always exhibit temporal dependency. If a packet i is lost, then packet $i + 1$ is likely lost too, leading to bursty losses and late losses. The correlation between delay and loss was investigated in [21] and [22].

2.2.1 Gilbert Packet Loss Model

Unconditional loss probability P (packet n is lost) has great impact on the performance of end-to-end interactive applications [15]. However, packet loss characteristics require study of the loss patterns, in particular, the correlation between consecutive packets. The Gilbert packet loss model [23] captures the conditional probability of the next packet's state (loss or received) with respect to the previous packet's state. Measurement of unicast and multicast traffic in the Internet indicates that the probability of loss of k successive packets decreases geometrically [24], hence Gilbert packet loss model is a well-fit candidate. The state diagram of the model is shown in Figure 2.3. Let us define 1 be packet loss and 0 be packet received. For the i^{th} packet,

$$p = P(X_i = 1 | X_{i-1} = 0)$$

$$q = P(X_i = 0 | X_{i-1} = 1)$$

The unconditional loss probability is obtained as

$$P(X = 1) = \frac{p}{p + q}$$

The probability of having k consecutively lost packets once we observe a loss packet is geometrically distributed.

$$p_k = (1 - q)^{k-1} \cdot q$$

Deriving the model parameters from empirical data is introduced in [16].

$$p = \sum_{k=1}^{\infty} \frac{o_k}{a}$$

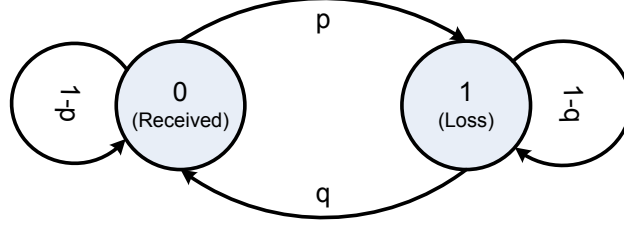


Figure 2.3: Gilbert Packet Loss Model

$$1 - q = \frac{\sum_{k=1}^{\infty} (k - 1) \cdot o_k}{\sum_{k=1}^{\infty} k \cdot o_k - 1}$$

where o_k is the number of loss episodes of length k , and a is the total number of packets sent.

The Gilbert model has a memory of only one past event. Work also has been done to extend the basic Gilbert model to have memory of past m loss events [16].

2.2.2 K^{th} Order Markov Chain Loss Model

The K^{th} Markov chain model has a memory of all the past k events [17]. The probability that the next event will be either a successfully received or a lost packet depends on the past k packets, regardless of the state of these k packets. The drawback of the K^{th} order Markov chain model is that all the last n states should be remembered. Since these events can have a value of 1 or 0 (loss or non-loss), 2^k states are needed to track the history of k events. Figure 2.4 shows the state diagram for $k = 2$.

The transition probabilities for the K^{th} order Markov chain model can be derived from the empirical data by the following formula [17]:

$$P(X_i = a | X_{i-1} = b_1, X_{i-2} = b_2, \dots, X_{i-n} = b_n) = \frac{n_{b'a}}{n_{b'}}$$

where b' is the state that $X_{i-1} = b_1, X_{i-2} = b_2, \dots, X_{i-n} = b_n$, $n_{b'}$ is the number of occurrences of state b' , and $n_{b'a}$ is the number of transitions from state b' to state a .

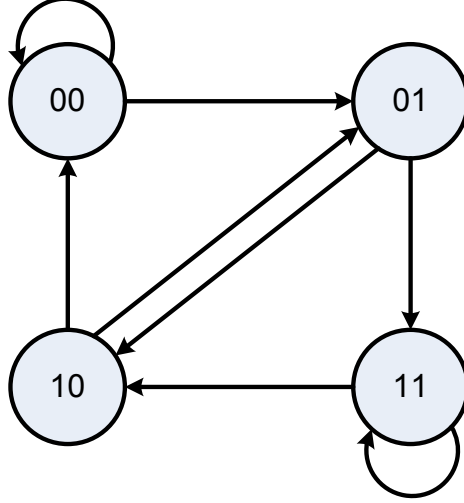


Figure 2.4: K^{th} Order Markov Chain Packet Loss Model, $K=2$

2.3 Ethernet Models

Ethernet is the world's most pervasive networking technology. The standard Ethernet uses CSMA/CD technology for multiple hosts to share the physical medium. A detailed CSMA/CD model with respect to the IEEE 802.3 specification observed in physical networks is described in [25]. The nodes in the simulated network compete for the shared medium. Once the medium is idle, one has to wait for an additional inter-frame spacing before starting data transmission. When a collision is detected, all the transmitting nodes perform a binary exponential backoff. After a predefined number of retransmission failures, the frame is dropped. The IEEE 802.3 specification with the CSMA/CD option is discussed in [26].

Switched Ethernet guarantees a dedicated bandwidth to each attached device. Gigabit and 10 Gigabit Ethernet use this technology rather than the traditional CSMA/CD technology. Network performance such as utilization and throughput is improved in the switched Ethernet. Data flow is point to point instead of broadcasting to every attached host, and thus generates much less traffic. Collisions may still occur if frames with the same target arrive at same time, but they are much less frequent than in the standard Ethernet.

Both the standard Ethernet and the switched Ethernet models were implemented in Real Time Immersive Network Simulation Environment (RINSE)

[27]. Model details are covered in Chapter 5. The switch models we built are intended to be used for Gigabit Ethernet simulation, but they can still be used on the standard Ethernet by simply specifying a different underlying MAC layer protocol in the simulator.

2.4 RINSE Network Simulator

RINSE is a computer network simulator, potentially at a large scale with tens of thousands up to millions of network entities [27]. RINSE also has an open and scalable emulation infrastructure to communicate with real network components, such as end-hosts and routers [28], and to interact with real applications in real time. Figure 2.5 shows the architecture of RINSE in the left and a typical virtual host in the right. The simulator is built on top of the Scalable Simulation Framework (SSF) [29], which is a standard or API for discrete-event simulation running on parallel platforms. Above SSF is the SSFNet, which includes various network components, such as host, router and link, and it supports a range of implemented network protocols, such as Ethernet, TCP, UDP, Open Shortest Path First (OSPF), Border Gateway Protocol (BGP) and Distributed Network Protocol (DNP3). Domain Modeling Language (DML) at the top is a scripting language, which can be used to easily create network topology and configure hosts, protocols, links and traffic [30].

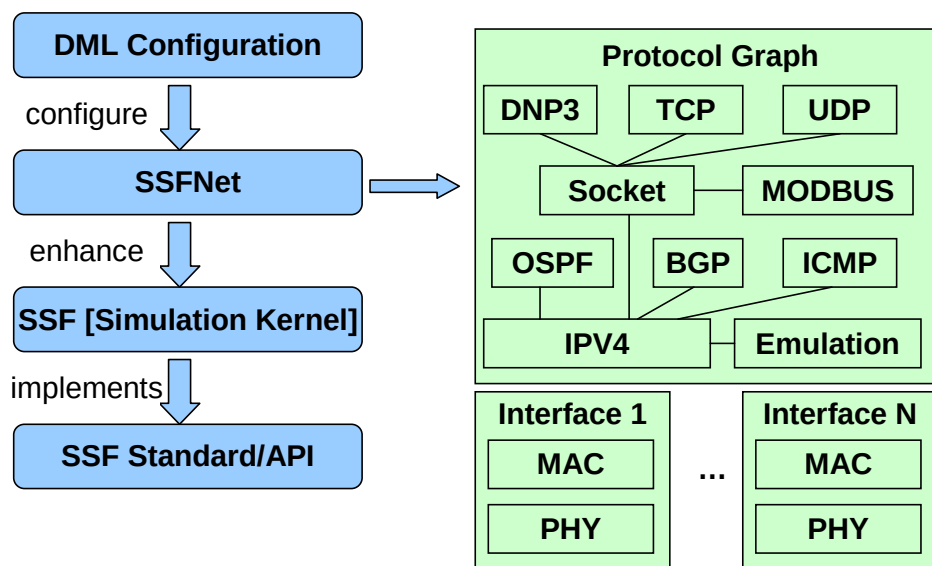


Figure 2.5: RINSE Network Simulator Architecture

CHAPTER 3

MEASUREMENT

3.1 Requirement

A comprehensive study of frame loss and delay patterns in a Gigabit Ethernet switch requires a testbed to

- generate traffic up to line rate with user configured parameters such as frame size, sending rate and inter-frame gap,
- record frame delays and arrival orderings with microsecond resolution,
- capture frames at line rate with little loss.

3.2 Testbed

We initially tried to use software-based time stamps, and the closest point to the physical medium is the NIC driver [31]. Two Intel E1000 drivers were modified to add time stamps into data payload upon a frame leaving a NIC and a frame arriving at another NIC. Time stamps were derived from CPU ticks. A new API (NAPI) mode was also enabled in the NIC driver to avoid the extra delay introduced by the interrupt mode. Two identical constant bit rate (CBR) UDP flows were sent from a sender to a receiver, one through a switch and the other through a wire connecting two NICs. The difference is the delay introduced by the switch based on the measurement that the latency in a wire is extremely small with little variance. The results show that under low sending rate (< 200 Mb/s), we can collect a sequence of frame delays with microsecond resolution and little noise was observed. However, the measurement noise added by buffering and processing overhead in NICs and stacks under high bit rate (> 500 Mb/s) was at least an order of

magnitude larger than the true delays. With such a large amount of noise, it is hard to recover frame delay in the switch.

As we were unable to obtain desirable accuracy through the instrumentation, we built a testbed that uses hardware to instrument, transmit, and capture Ethernet frames at line rates. Figure 3.1 is the overview of our first testbed. The testbed is composed of a PC with four dual-core 2.0 GHz CPUs running CentOS 5.2, a NetFPGA card and a switch connected by cat-6 Ethernet cables. Traffic is generated using a 4-port NetFPGA card [32], [33]; the frames to send can be loaded from a pcap file with a specified sending rate. The obvious way to measure delay is to time-stamp a frame at transmission and after passage through the switch, but this approach has its own challenges, not least of which is that our traffic generator could not easily generate time stamps! Even if it could, we would have to synchronize the clock on the NetFPGA card with the clock on the receiver. In order to time the passage of a frame through a switch, we took advantage of the NetFPGA card’s ability to simultaneously send identical flows from each port. Once two identical flows are generated from the NetFPGA to the same destination — one through wire and the other through the switch — the difference of the two receiving time stamps is the delay through the switch. For example, in Figure 3.1 a frame is sent simultaneously out on ports 1 and 3 of the NetFPGA card. One instance arrives at port 2 of the NetFPGA and is time-stamped on receipt. The other enters the switch via port 2, is routed out via port 1, and arrives at the NetFPGA on port 4 where it is time-stamped. The difference in time stamps is the delay through the switch (and time on the wire from switch to NetFPGA). To validate the approach, we replaced the switch with a wire and calculated the difference under a 1 Gb/s sending rate. The measured delay has mean 0 ns and standard deviation 0.004 ns, which is low enough given the microsecond delay in the switch. By utilizing all four ports of the NetFPGA card, the testbed can monitor one flow in real time. However, the 1 Gb/s line rates were too much for this approach to provide the desired capture rate. With the current version of NetFGPA packet generator, we can only capture a few hundred frames at line rate. By using the high speed traffic capture tool, such as gulp [34], we can improve the capture rate to 2000 continuous frames with zero loss, which is still not long enough to reveal the long-term behavior of a switch.

Then we integrated the Endace DAG [35] to build the second testbed.

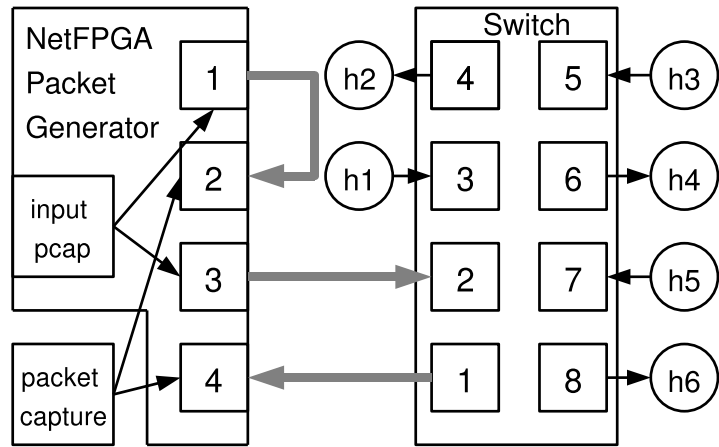


Figure 3.1: Testbed One Setup

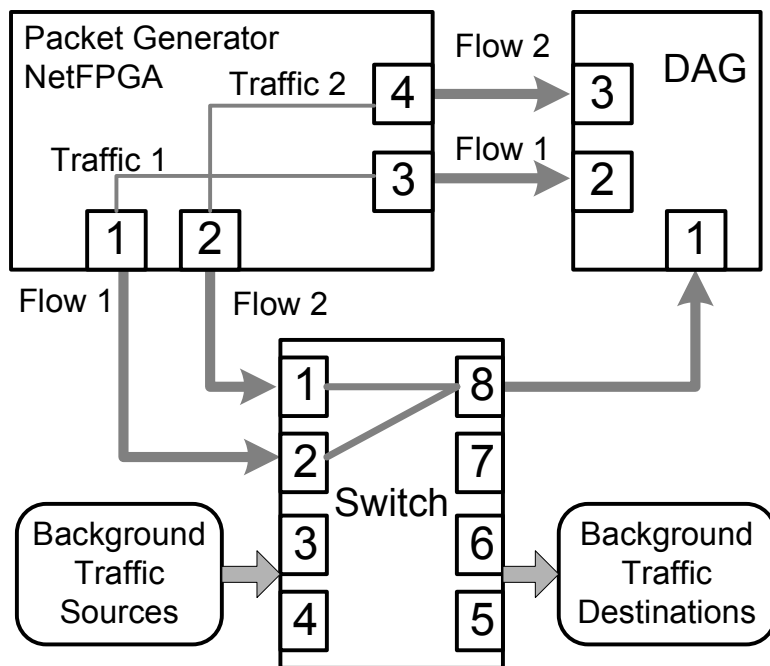


Figure 3.2: Testbed Two Setup

Figure 3.2 depicts our solution. The Endace card stamps received frames with a clock having a 10 ns resolution; the card can also capture and store millions of frames with zero loss at 1 Gb/s. By utilizing all four ports of the NetFPGA card and two ports of the DAG card as shown in Figure 3.2, the testbed can monitor two flows in real time. Both cards are placed on the same PC. We captured data from three 8-port Gigabit Ethernet switches: 3COM 3CGSU08, NetGear GS108 and TrendNet TEGS80G. They are all simple low-end commodity-use switches with no configuration interface available. All ports were connected by cat-6 Ethernet cables.

The data flows generated in the experiments were constant bit rate (CBR) Ethernet raw frames. We generated traffic files in pcap format, and added a sequence number into each frame. The NetFPGA card transmits frames as specified in the pcap file out on two different ports. The copy which travels immediately to the DAG card is always received; the copy that passes through the switch might possibly be dropped. Post analysis of the received frames thus identifies the missing frames. Likewise the difference between time stamps on frames with identical sequence numbers (one which passed directly to the DAG, the other which passed first through the switch) gives that frame’s delay. The frame size is fixed at 1500 bytes in order to achieve maximum sending/receiving rate by minimizing the effect of the inter-frame gap. We generated one million frames for the “Traffic 1” flow, and also for the “Traffic 2” flow.

While we performed experiments on three different switches, the behaviors of the TrendNet and NetGear switches were very similar in all cases. Correspondingly, the rest of the paper will only show results from the NetGear and 3COM switches.

CHAPTER 4

ANALYSIS AND MODELING

4.1 Data Analysis

The first set of experiments monitored a single flow whose sending rate varies from 100 Mb/s to 1 Gb/s with 100 Mb/s increment, and no background traffic is generated. The idea is to establish a baseline delay, in a context where there is no contention, and the input rate is no greater than line rate. Figure 4.1 shows the frame delay of the monitored single flow collected from both switches. The delay in both switches behaves constantly with no loss. The delay has the same mean value under any sending rate up to 1 Gb/s and the variance is close to 0 μ s. We learned from this that the switches indeed handle line rate without loss, the switches have significantly different delays, and that with deterministic inter-arrival times those delays are constant. In addition, we also varied the packet size from 64 byte, which is the minimum Ethernet frame size, to 1500 bytes, which is the maximum Ethernet frame size. Figure 4.2 shows that packet delay increases linearly as the packet size increases for both switches. Therefore, the packet delay for a single flow can be modeled by a straight line with respect to the packet size.

In the second set of experiments we kept the single flow, and added various combination of background traffic flows going through other ports, such as three parallel 1 Gb/s CBR flows and five 1 Gb/s CBR flows from five input ports to one output port (see again Figure 3.2), *other than* the one targeted by the monitored flow. These experiments revealed the same behavior of delay and loss on the monitored flow as we saw in the first experiments when there were no background flows. From these experiments we learned that both switches can handle nearly 1 Gb/s flows at each output—no frame losses were observed until the input rate reached 987 Mb/s. We are secure in using the constant value shown in Figure 4.1 as the baseline non-queuing

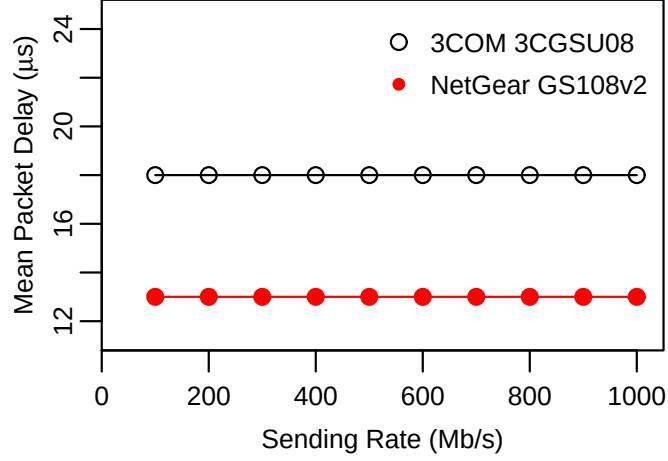


Figure 4.1: Frame Delay for Single Flow Traffic

delay added by a switch.

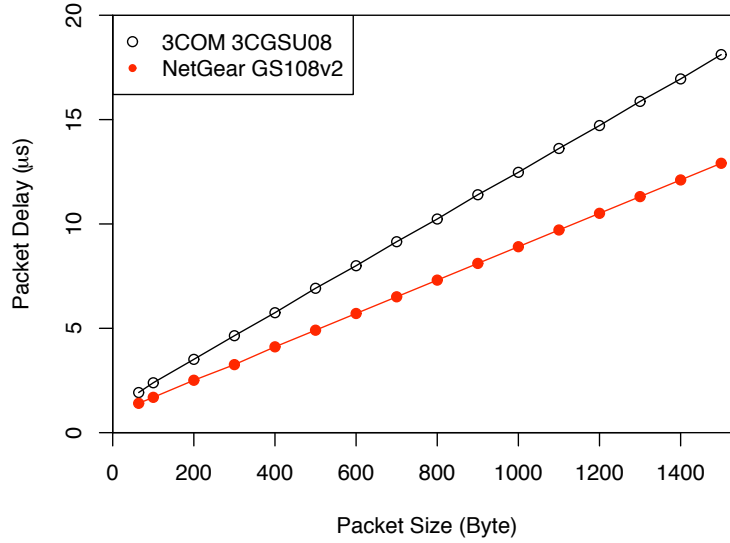


Figure 4.2: Frame Size vs. Packet Delay for Single Flow Traffic

The third set of experiments monitored two flows (“Traffic 1” and “Traffic 2” in Figure 3.2) from two input ports to the same output port; we varied the sending rates on both flows from 100 Mb/s to 1 Gb/s with a 100 Mb/s increment. Traces were completely collected at the DAG card and post-processed to compute delay and loss patterns. When total input rate of the two injected flows did not exceed the switch’s service capacity (which is 1 Gb/s in theory and 987 Mb/s as observed), both switches experienced

small delay and zero loss. The delay is at most two times the minimum delay as shown in Figure 4.1, which means at most one early frame is queued upon arrival and the switch is not congested. This is valuable information, particularly when we are able to tolerate a 100% error in frame latency—if the output port is not congested, we can model the delay through the switch with a constant, and see no frame loss.

The more interesting and challenging case is of course when the total input rate exceeds the maximum service rate; both switches experienced large delay and loss. However, the two switches have significantly different delay and loss patterns. Figure 4.4(a1) and (b1) (see page 22) show one sample pattern of two input flows with sending rates 900 Mb/s and 300 Mb/s—4.4(a1) describes the 3COM switch and 4.4(b1) describes the NetGear switch. Frames are ordered according their arrival timestamp. The delay is plotted on the y-axis and a value zero represents a loss.

A very interesting difference is that the NetGear switch dropped frames from both 900 Mb/s and 300 Mb/s flows, in bursts, while the 3COM switch dropped frames only from the 900 Mb/s flow, not in bursts. Indeed, as we increased the rate of the slower flow, the 3COM switch did not drop any frames from it until it reached 500 Mb/s. The delays of both flows increase together and stabilize at same level for the NetGear switch, while the delay of the two flows stabilized at different values in the 3COM switch.

The fourth set of experiments monitored the output rate of three flows from three input ports to the same output port. We fixed the sending rate of flow 1 to 100 Mb/s and the sending rate of flow 2 to 500 Mb/s, and varied the sending rate of flow 3 from 100 Mb/s to 1 Gb/s with a 100 Mb/s increment. Table 4.1 shows the results from the NetGear switch and Table 4.2 shows the results from the 3COM switch. When the sum of the input rate does not exceed the switch’s service capacity, every flow gets through the switch without losing a frame. When the total input rate exceeds the maximum service rate, NetGear drops frames in all the three flows and the drop rate is almost same as shown in Table 4.3. For the 3COM switch, we observed that (1) frames in flow 1 (100 Mb/s) never got dropped, and (2) frames in flow 2 and 3 (≥ 500 Mb/s) equally share the remaining bandwidth. This implies that all flows had equal bandwidth reservation and the switch served the connections proportionally to their reservation and then distributed the extra bandwidth equally among the active flows.

Table 4.1: NetGear, Output Rates, Three Flows from Three Input Port to One Output Port

Input Rate (Mb/s)			Output Rate (Mb/s)		
Flow 1	Flow 2	Flow 3	Flow 1	Flow 2	Flow 3
100	499	100	100	499	100
100	499	200	100	499	200
100	499	299	100	499	299
100	499	399	98	492	394
100	499	499	90	447	447
100	499	599	83	414	487
100	499	699	77	383	525
100	499	799	72	359	554
100	499	899	68	339	578
100	499	988	65	316	604

Table 4.2: 3COM, Output Rates, Three Flows from Three Input Port to One Output Port

Input Rate (Mb/s)			Output Rate (Mb/s)		
Flow 1	Flow 2	Flow 3	Flow 1	Flow 2	Flow 3
100	499	100	100	499	100
100	499	200	100	499	200
100	499	299	100	499	299
100	499	399	100	486	399
100	499	499	100	443	442
100	499	599	100	443	442
100	499	699	100	443	442
100	499	799	100	443	442
100	499	899	100	443	442
100	499	988	100	442	443

Table 4.3: NetGear, Drop Rate, Three Flows from Three Input Port to One Output Port

Input Rate (Mb/s)			Drop Rate		
Flow 1	Flow 2	Flow 3	Flow 1	Flow 2	Flow 3
100	499	100	0	0	0
100	499	200	0	0	0
100	499	299	0	0	0
100	499	399	0.02	0.01	0.01
100	499	499	0.10	0.10	0.10
100	499	599	0.17	0.17	0.19
100	499	699	0.23	0.23	0.25
100	499	799	0.28	0.28	0.31
100	499	899	0.32	0.32	0.36
100	499	988	0.35	0.37	0.39

From these experiments we learn that

- an accurate model needs to be aware of differences between switches,
- an accurate model needs to use parameters derived from experimental data,
- loss patterns indicate that something very different is happening inside the two switches, and we ought to be able to explain it.

4.2 Queueing Switch Models

The loss and delay pattern observed in the NetGear switch can be well explained by the FCFS output queue as shown in Figure 4.3(a), in which the two flows can be effectively replaced by one flow with the aggregated sending rate. The large loss episode observed can be explained by a policy that once loss occurs, the switch will continue to drop further incoming frames until the queue length drops below a threshold Q_R . The device-specific parameters of this model include the service rate R_S , the queue size Q_S and Q_R . From the experimental data of NG, we estimated $R_S = 987$ Mb/s, $Q_S = 22$ frames and $Q_R = 11$ frames.

Behavior of the 3COM switch can be explained by an architecture (see Figure 4.3(b)) where frames are in queues associated with their input ports

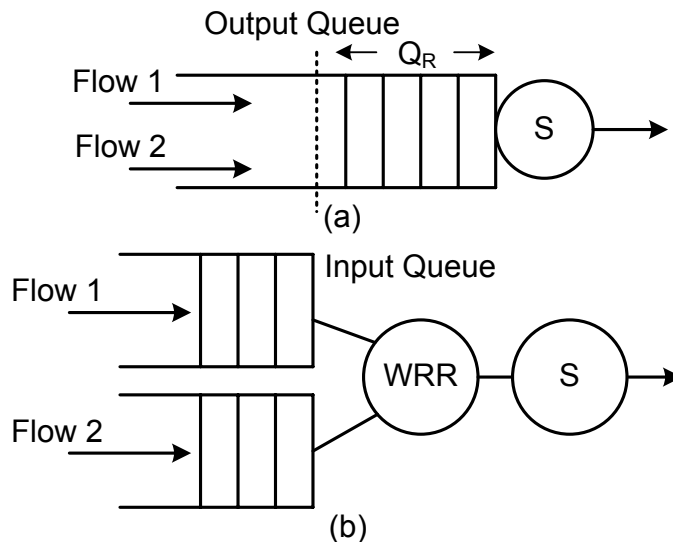


Figure 4.3: (a) Output Queue Model for NetGear GS108 (b) Input Queue Model with WRR for 3COM 3CGSU08

(or possibly queues for individual flows), and some scheduling algorithm is used that gives priority to input queues based on weights that are proportional to flow rates or queue length. Such a scheduling algorithm could give service to the slow flow often enough that it does not drop frames until it requires more than half the bandwidth; frames in the slower flow could have smaller latencies than frames in the fast flow because the queue is smaller and enough attention is given to the slow queue. A number of different policies might be at play here, particularly those that provide *rate proportional service* [2]. In the **general case** the choice of next frame to transmit is a function of the whole switch state at the time of the decision, and is made for each frame. This means that **in general** the time at which a frame is served is not predictable. This in turn implies that *at a minimum* there are two events per frame—its arrival, and its departure (because the departure time cannot be determined at the frame’s arrival.) A notable exception to the general case is FCFS of course.

We do not know exactly what policy is implemented within the 3COM switch to achieve equal bandwidth reservation for every flow; from among the potential service disciplines we investigated use of the weighted round robin (WRR) scheduling policy [36]. Actually, many commercial switches are using round robin based scheduling due to the low time complexity $O(1)$ and

low implementation cost [37]. Under WRR each input queue is visited in a round-robin fashion, but queues may have different numbers of frames served each visit. One computes the number of frames to transmit from a queue i as $\lfloor Q_i/Q_{\min} \rfloor$, where Q_i is its queue length, and $Q_{\min}$ is the minimum queue length among all non-empty queues competing for service. For example, if $\lfloor Q_2/Q_1 \rfloor = N$, then N frames from queue 2 are served while one frame from queue 1 is served.

With a 900 Mb/s sending rate in flow 1 and 300 Mb/s in flow 2, the ratio of queue lengths should be approximately 3:1 in steady state, and the effective service rates are 700 Mb/s and 300 Mb/s respectively. Therefore, for the 300 Mb/s flow, no loss was observed and the stabilized delay should be smaller than that of the 900 Mb/s flow since its queue is not yet full. With this model, the low load flow will encounter loss once its sending rate is above 500 Mb/s, which matches our experimental data. From our data we estimated the same device-specific parameters for the 3COM switch to describe post-loss behavior : R_S is 987 Mb/s, Q_S is 9 frames and Q_R is $Q_S - 1$.

We developed discrete-event simulation models of both switches and repeated the third set of experiments using the traces as input to the switches, and used this queuing model to select lost frames and compute delay.

Figure 4.4(a2) and (b2) plot the delays corresponding to the same real switch test case as shown in Figure 4.4(a1) and (b1). Excellent—indeed, nearly perfect—agreement is found between simulation and real data on the NetGear switch model. The FCFS assumption appears to be well-founded. The agreement is also quite good between real data and simulated 3COM switch. The average delays for both flows match well, and the simulated model dropped frames only from the faster flow, and at the same rate as the faster flow. There is not a frame-by-frame matching as we observed in the NetGear switch model.

Our simulation model of the 3COM switch has two events per frame. One event occurs when the frame arrives, another when a selected frame completes transmission. Scheduler action is initiated either when a frame arrives (and the system is empty) or at the completion of a frame transmission, and so while it adds some computational weight to the events, there are but two events per frame. This implementation makes it straightforward to replace WRR with a different scheduling policy, and the execution cost of the

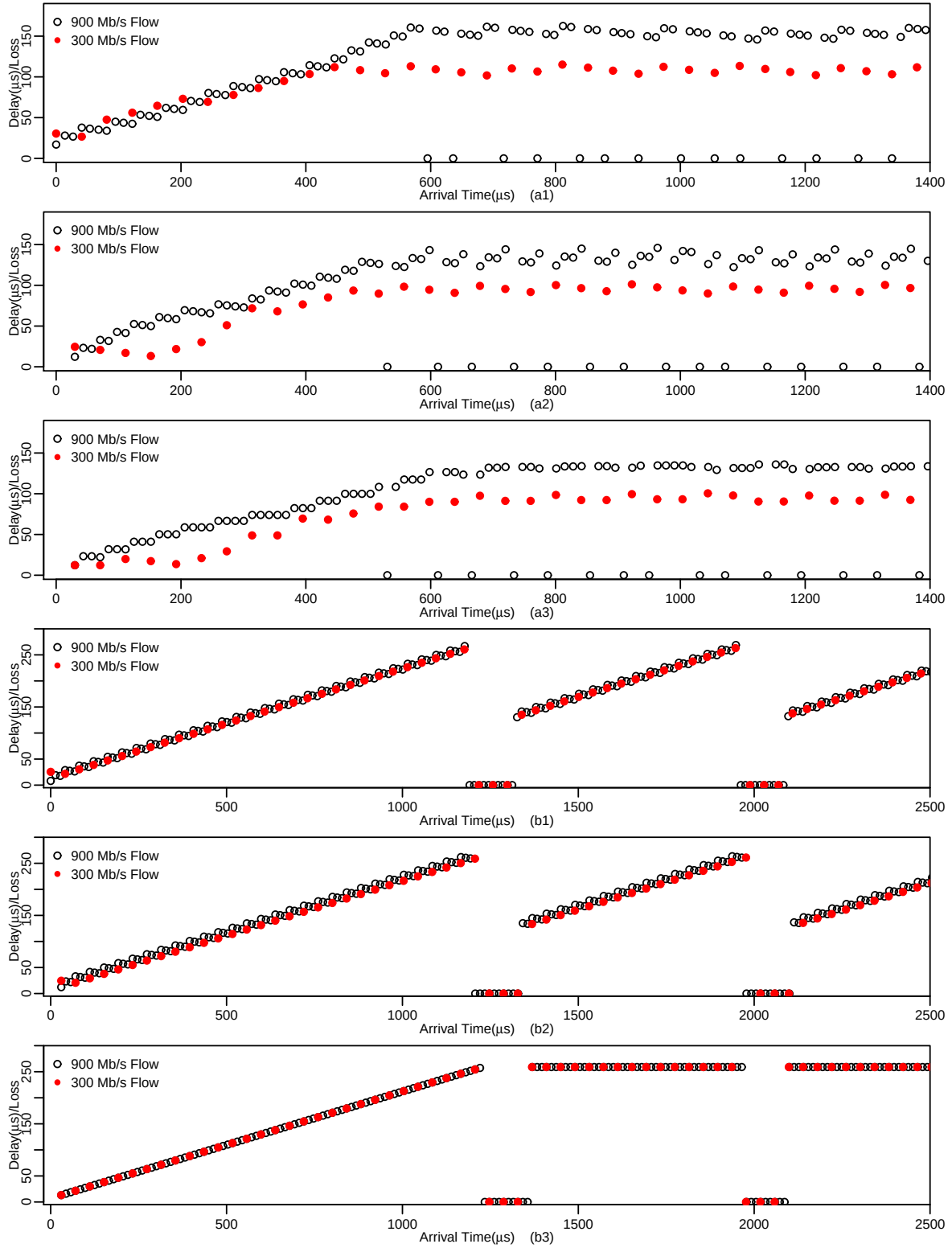


Figure 4.4: Delay/Loss Pattern, Two Flows to One Output Port, (a1) 3COM Real (a2) 3COM Queueing Model (a3) 3COM Simplified Queueing Model (b1) NetGear Real (b2) NetGear Queueing Model (b3) NetGear Analytical Model

implementation is a reasonable representation of switches that provide rate-proportional service. However, observe that under the structure of WRR, the departure times of all frames scheduled to be served in one scheduling round are known at the conclusion of the scheduling decisions. This makes possible an implementation where there is one event per frame arrival, and one event per scheduling round. Our performance analysis will include consideration of this optimization.

The simulation model of the NetGear switch can be implemented using one event per frame. On arrival, all the information needed to determine when the frame enters service and departs is known, and so its arrival at the next switch in the sequence can be scheduled immediately. This “one-event cost” property is possible only with FCFS scheduling.

4.3 Latency-Approximate Switch Models

In our drive to achieve high performance switch simulations we will at some point have to knowingly trade off accuracy of some attribute for gained speed. Of latency and loss, for our intended use we would sacrifice latency. The time scale at which applications run is at least two orders of magnitude slower than switches, so a factor of two error in switch latency is lost in the noise at that scale. Loss can trigger new behaviors in applications, and so we attempt to be as faithful to it as we can.

We now develop for both the NetGear and 3COM switches models that we call “Latency-Approximate” models. At one event per frame per switch the NetGear model is already efficient; our latency-approximate model for it is more in line with future work where latency and loss will be extracted more from flow rate characteristics than from frame-by-frame simulation. Still, it is appropriate here to develop the model and comment on its accuracy. Our latency-approximate model for the 3COM switch maintains accurate queue length state information, and can infer what the true latency for a frame is, but will estimate that latency *at the time of arrival* and will immediately schedule the arrival of the frame at the next switch. The latency estimate is based on recent *true* latencies in the recent past.

4.3.1 Simplified Aggregated FCFS with Draining

As we have seen, the NetGear data suggests an internal queueing architecture where flows with a common destination port are aggregated and served in FCFS fashion. A frame loss triggers a “draining” phase, where additional frames are dropped until the queue size reaches some threshold. Some sequence of frames are accepted, and then another loss triggers another draining stage. In addition, we observed in the NetGear data a transient period where the queue warms up to the congestion stage. We describe this pattern in Figure 4.5.

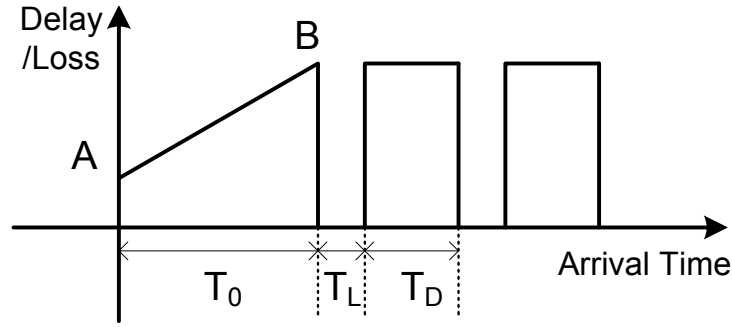


Figure 4.5: Simple Model

The pattern may be parametrized by the following variables:

A	Minimum latency, (see Figure 4.1)
B	Mean latency in the “warmed” stage
T_0	Time duration for first stage
T_L	Avg. time of loss episode
T_D	Avg. time between neighboring loss episodes
R_A	Aggregated arrival rate
R_S	Service rate of a switch
Q_S	Maximum queue size per port
Q_R	Required queue size for readmitting incoming frames after frame drop
Y	Delay/loss value, with delay > 0 and loss = 0

$$Y = \begin{cases} \frac{A+(B-A) \times t}{T_0} & t \in [0, T_0] \\ 0 & t \in (T_0 + n(T_L + T_D), T_0 + T_L + n(T_L + T_D)], \\ & \text{for some } n \in N \\ B & t \in (T_0 + T_L + n(T_L + T_D), T_0 + (n+1)(T_L + T_D)], \\ & \text{for some } n \in N \end{cases}$$

The idea is to keep track of where the output queue is in this pattern, and when an arrival occurs apply the equation for Y to determine whether it is lost, and if not, what its (constant) latency will be. Computationally this is simpler than queueing frames explicitly and computing precise latencies. It also fits in well with a flow-rate oriented formulation where input flow rates affect state variable R_A .

Figure 4.4(b3) shows the loss and delay generated by the analytical model when used as the basis of a simulation driven by the gathered traces. Compared with the real trace in Figure 4.4(b1) we see that the model accurately captures the loss rate, the length of the loss episode, and the time between successive loss episodes. We could easily modify the formula for latency to estimate queue length at the time of an arrival and fine-tune the latency estimate correspondingly. We decided (perhaps arbitrarily) to use a constant, looking ahead to the future use of this model we have already mentioned.

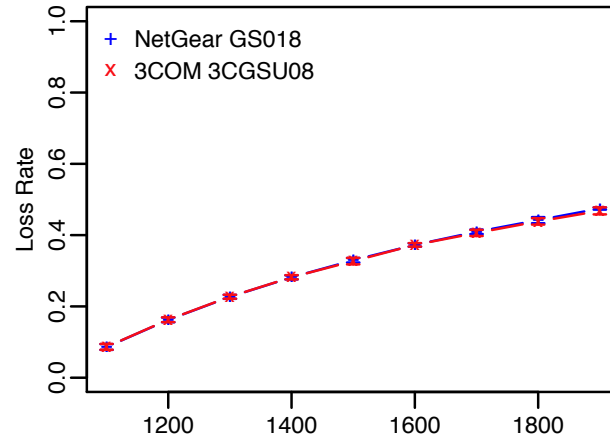
The model is designed from the switch’s viewpoint with delay and loss calculated as per output port. By aggregating the flows directed to the same output, the model is greatly simplified for speed gain. To validate the idea of aggregating input flows, we grouped the results with the same aggregated sending rates from the third experimental set and calculated the mean and standard deviation for loss-related metrics, including loss rate and average time of loss episode T_L . As shown in Figure 4.6(a) and (b), both switches have small standard deviation for the two metrics. Figure 4.6(a) shows that the loss rate increases linearly with R_A , which is a strong indication that the model can be generalized to all sending rates using linear interpolation. By further investigating the T_L and T_D , we found that T_L is constant for any R_A in Figure 4.6(b), and $T_D \times (R_A - R_S)$ is also constant in Figure 4.6(c), which means $T_D \propto 1/(R_A - R_S)$. It is because

$$\begin{aligned} T_L &\approx (Q_S - Q_R)/R_S \\ T_D \times (R_A - R_S) &\approx (Q_S - Q_R) \\ \text{Similarly, } T_0 \times (R_A - R_S) &\approx Q_S \end{aligned}$$

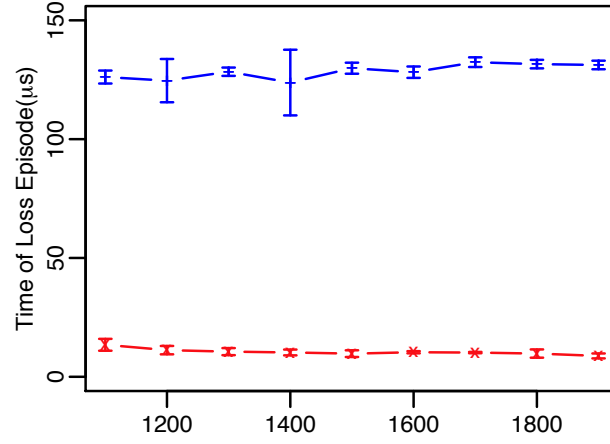
For a particular switch, Q_S, Q_R, R_S are fixed. Since T_L is a constant, T_D and T_0 can be computed by linearly interpolating Figure 4.6(b) and (c) for any given R_A ; therefore, all the required parameters for the analytical model can be derived for a given R_A , and the model can be presented as a function of R_A .

4.3.2 Latency-Approximate WRR

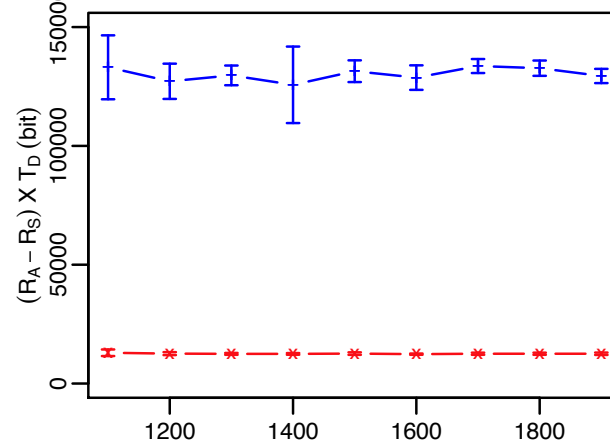
The latency-approximate model of a WRR managed switch will faithfully maintain queue state of all input queues, and so when a frame arrives it can faithfully determine whether a frame arriving at the instance would be dropped. At the arrival time it also *estimates* a latency, which enables one to schedule the arrival of that frame at the next switch or host immediately. This has the obvious advantage of avoiding a later “frame departure” event, but also has the perhaps not-so-obvious benefit of creating a larger temporal separation between the switch and the time stamps on the events it forwards. This larger temporal separation has a benefit in paral-



(a)



(b)



(c)

Aggregated Sending Rate (Mb/s)

Figure 4.6: Analytical Model Validation

lel simulation—that event passing through that output port with that time stamp is a promise to its recipient that the switch will not post another event through that port with smaller time stamp. Exactly this kind of information is key for many conservative synchronization strategies. Finally, the latency-approximate switch model has a lower overhead simply by not managing the explicit queueing of frames.

WRR works in rounds. At the scheduling point of each round, the number of frames from each queue that will be served in that round is computed. It is possible to determine the state of every queue at all points during the upcoming round except for arrivals. We can compute the length of the queue upon new arrivals and hence accurately decide whether it is queued or dropped. The algorithm updates the following state variables when a frame arrives, and when a scheduling round executes:

$N_{s,j}(t)$	#frames scheduled at j , but not sent by time t
$N_{u,j}$	# frames left unscheduled at j
$N_{a,j}$	# frames arriving at j after previous round
Q_j	# frames in queue j
$Q_{max,j}(t)$	threshold at j for accepting frames, at time t
Q_{min}	Min. # frames among all non-empty queues
S	service time for a frame

Executed when frame i arrives at input queue j at time t :

$$Q_j = N_{s,j}(t) + N_{u,j} + N_{a,j}$$

if $Q_j \geq Q_{max,j}(t)$ **then**

Drop frame i

else

Estimate delay d

Schedule arrival of frame i at next switch, d time in future

$N_{a,j}++$

end if

if server is idle **then**

Start the scheduling round

end if

Executed as the scheduling round:

if $\exists j$ s.t. Q_j is not empty **then**

```

for all queues  $j$  do
   $N_j = \lfloor Q_j / Q_{min} \rfloor$ 
  Record departure times of the  $N_j$  frames
   $N_{u,j} = Q_j - N_j$ 
   $N_{a,j} = 0$ 
end for
Schedule the next round at time  $S \sum N_j$ 
end if

```

In the above algorithm, some computation is required to determine $N_{s,j}(t)$, based on t and the saved list of departure times of scheduled frames. Likewise, $Q_{max,j}(t)$ varies between the maximum queue length and Q_R , depending on whether the queue is in the draining state (triggered by a loss) or not.

A scheduling round is activated either when a frame arrives and the server is idle or when the current scheduling round finishes and there are still frames waiting to be served. For correct modeling of loss, all we need to keep track of is the queue length using a few counters; in particular it is unnecessary to queue and dequeue the frames.

One interesting point is that at the time of a frame's arrival we can determine whether it is dropped or not, but we cannot determine exactly when the frame will be served, because that time is dependent on future arrivals up until the time of the next scheduling round. However, we can create an estimate that is accurate in a statistical sense by utilizing the historical delay information computed at each scheduling round.

Define M_j to be the exponentially decayed average latency of frames in queue j as follows. Suppose that N_j frames are scheduled for transmission at queue j , and for $j = 1, 2, \dots, N_j$, let $L_{k,j}$ be the delay of k^{th} of these. At each scheduling round,

$$M_j = \alpha(1/N_j) \sum (L_{1,j} + L_{2,j} + \dots + L_{N_j,j}) + (1 - \alpha)M'_j$$

where M'_j is the latency average for queue j last computed by this formula. When a frame arrives at queue j , M_j is used as the latency estimator. A large value of α is desirable to make the latency estimator responsive to queue size.

Figure 4.4(a3) shows the simulation results for the same test case in Figure 4.4(a2) using the 3COM latency-approximate model with $\alpha=0.9$. The simplified model generates frame loss patterns as accurate as those of the

detailed model, with no losses in the low load flow and the same loss rate in the high load flow. With respect to delay, both flows stabilized near the values seen in the real data and in the detailed model, but the fluctuation is slightly less than what is observed in the real data, as one would expect from this smoothing.

4.4 Markov Chain Frame Loss Model

Frame losses were observed inside both switches when the load was higher than the maximum service rate. It was observed in the NetGear switch that once a frame of a data flow got lost, the next few frames were always lost too. Then after a few received frames, frame losses appeared again. Such loss and receiving patterns were always presented alternatively for some rounds before losses disappeared for long duration. Let us define 1 for a lost frame and 0 for a received frame. Figure 4.7 shows a sample pattern of frame loss. The result indicates that strong autocorrelation of frame loss exists among neighboring packets. Therefore, we want to design a good frame loss model to capture both loss rate and autocorrelation regardless of the switch scheduling policy in use.

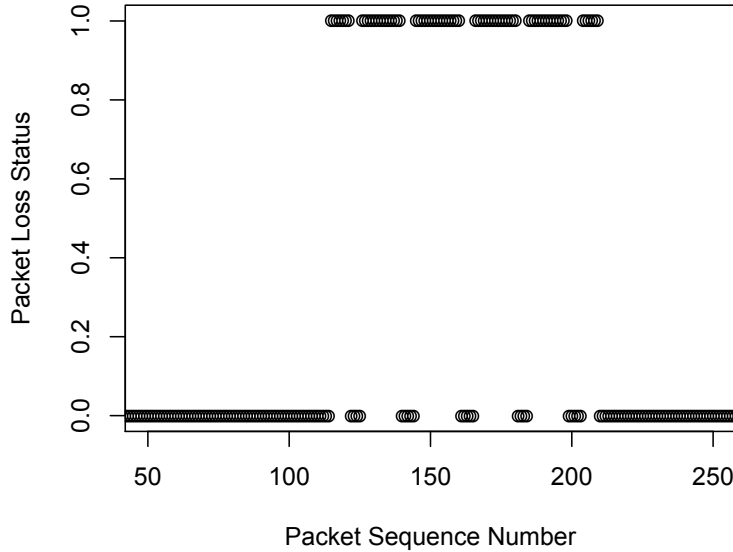


Figure 4.7: Packet Loss in Time Sequence (0 - Received, 1 - Loss)

Finite state Markov processes can capture a large variety of temporal de-

dependencies and thus are used widely to model frame loss; these models include the Gilbert model, the extended Gilbert model [16] and the K^{th} order Markov chain model [17]. In the Markov chain model, the current state of the process depends on a certain number of previous values of the process. The Gilbert model has a memory of only one past event. The extended Gilbert model ignores the dependency of successive received frames. The K^{th} order Markov chain can capture the entire dependency of K successive frames. However, due to the strong autocorrelation among frames, a large k has to be used, which results 2^k states. Large state space adds great complexity to the model, so it is not desired for large-scale simulation. Therefore, we propose a Markov chain model based on the observed frames loss patterns, which has a much smaller state space than K^{th} order Markov chain and still accurately models the autocorrelation of both received and lost frames.

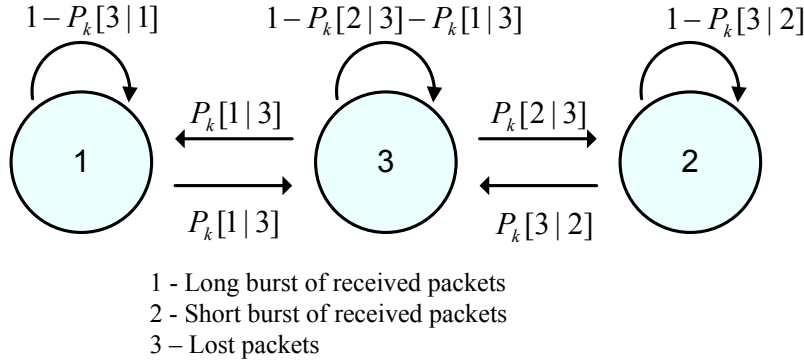


Figure 4.8: State Diagram of Packet Loss Model

We define three types of states in our model. Type 1 represents a long burst of received frames, type 2 represents a short burst of received frames, and type 3 represents lost frames. The next state depends on the current state and the number of successive frames already seen in the same type of state.

$$\begin{aligned}
P[X_i = s' | X_{i-1} = X_{i-2} = \dots = X_{i-k} = s] \\
= P_k[s' | s] \text{ where } s, s' \in \{1, 2, 3\}
\end{aligned}$$

Figure 4.8 shows the state diagram of our model and Figure 4.9 shows the expanded state diagram for M consecutive received frames in long burst, K consecutive received frames in short burst and N consecutive lost frames.

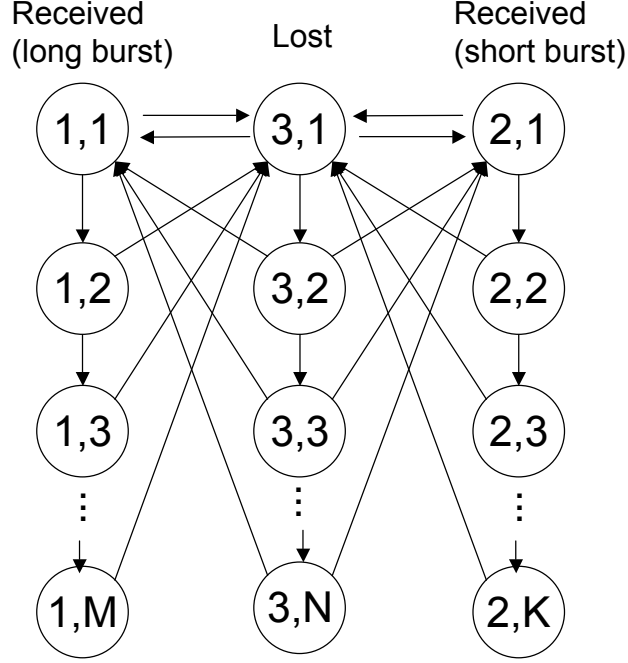


Figure 4.9: Expanded State Diagram of Packet Loss Model

Since received frames and lost frames occur alternatively, no transition should occur between type 1 state and type 2 state. Let $\{x_i\}_{i=1}^n$ be an observed sequence from a Markov source. The state transition probabilities of the Markov chain can be estimated as follows.

Let $b^k = (b_1, b_2, \dots, b_k), b \in \{1, 2, 3\}$ be a given state of the chain. Let $n_{b^k a}$ be the number of times state b^k is followed by value a in the sample sequence. Let n_{b^k} be the number of times state b^k is seen. Let $p_{b^k \rightarrow a}$ be an estimate of the probability that $x_i = a$, given that $x_{i-k} = \dots = x_{i-1} = b$. Then $p_{b^k \rightarrow a}$ estimates the state transition probability from state b^k to state $(b_1, b_2, \dots, b_k, a)$. The maximum likelihood estimators of the state transition probabilities of our model are

$$p_{b^k \rightarrow a} = \begin{cases} \frac{n_{b^k a}}{n_{b^k}} & \text{if } n_{b^k} > 0 \\ 0 & \text{otherwise} \end{cases}$$

The probability transition matrices computed from the 20 traces collected from the NetGear switch are very close, which is a strong evidence to show that the assumed Markov property in our packet loss model exists.

4.5 Multivariate Gaussian Autocorrelation Model

We also investigate the autocorrelation of neighboring frames, which is one level beyond modeling the marginal distribution of delay. Experimental results in Table 4.4 indicate that the delay process $\{X_t\}$ has strong autocorrelation. Therefore, our delay model has to efficiently generate a time series of delay which captures the observed autocorrelation while still matching the empirical marginal distribution. Related modeling and time series generation techniques include ARTA-like model [38], VARTA [39], NORTA [40], TES [41] and a family of Copula models [42]. Gaussian Copula was selected to build our delay model due to its computation efficiency and accuracy in modeling autocorrelation.

Table 4.4: Autocorrelation of the Delay Processes $\{X_t\}$ and $\{W_t\}$

Lag	$\{X_t\}$	$\{W_t\}$
0	1.000	1.000
1	0.998	0.947
2	0.996	0.940
3	0.994	0.945
4	0.992	0.932
5	0.990	0.923
6	0.988	0.917
7	0.985	0.907
8	0.981	0.890
9	0.979	0.882

A copula is a multivariate joint distribution whose marginal distributions are all uniformly distributed on the interval $[0,1]$. Let $F_{X_1, X_2, \dots, X_n}$ denote the joint distribution of n random variables and C denote the corresponding copula. The Sklar's theorem [43] shows that a joint distribution can be characterized by two components: (1) the individual marginal distributions, and (2) the copula which captures the entire dependency structure.

$$F_{X_1, X_2, \dots, X_n}(X_1, X_2, \dots, X_n) = C(u_1, u_2, \dots, u_n)$$

with the marginal distribution

$$F_{X_1}(X_1) = u_1, \dots, F_{X_n}(X_n) = u_n \sim Uniform(0, 1)$$

Gaussian copula is constructed from the multivariate Gaussian distribution via the Sklar's theorem. Letting Φ be the standard multivariate Gaussian cumulative distribution function with correlation coefficient matrix ρ , the Gaussian copula function is

$$C_\rho(u_1, u_2, \dots, u_n) = \Phi_\rho(\Phi^{-1}(u_1), \Phi^{-1}(u_2), \dots, \Phi^{-1}(u_n))$$

Our delay model utilized the Gaussian copula to generate a time series with specified autocorrelation and marginal distribution derived from experimental data. The generation procedures are as follows:

Assume that the frame delay in a switch is a stationary time series, and the marginal distribution of each frame delay is identical, and the maximum lag of the autocorrelation we want to capture is n .

1. Transform the empirical delay process $\{X_t\}$ to a standard Gaussian process $\{Y_t\}$ by $Y_t = \Phi^{-1}(F(X_t))$, where $F(X_t)$ is the marginal empirical CDF, and $\Phi(Y_t) \sim N(0, 1)$ is the standard Gaussian CDF.
2. Calculate the autocorrelation R of $\{Y_t\}$ for each lag up to n , and construct the (auto)correlation coefficient matrix ρ of the Gaussian copula.

$$R(k) = \frac{1}{(n-k)\sigma^2} \sum_{t=1}^{n-k} [Y_t - \mu][Y_{t+k} - \mu] = \frac{1}{(n-k)} \sum_{t=1}^{n-k} Y_t Y_{t+k}$$

ρ is a symmetric $(n+1) \times (n+1)$ matrix, where

$$\rho_{ab} = \begin{cases} R(|a-b|) & a \neq b \\ 1 & a = b \end{cases}$$

3. Generate a standard Gaussian time series $\{Z_t\}$ based on ρ , so that $\{Z_t\}$ has the same autocorrelation as $\{Y_t\}$. The generation of the next element Z_j is always conditioned on previous n elements $Z_{j-1}, Z_{j-2}, \dots, Z_{j-n}$. Since conditional distribution of multivariate Gaussian process is still Gaussian with the following mean $\tilde{\mu}$ and correlation coefficient matrix $\tilde{\rho}$, Z_j is generated by sampling the new Gaussian distribution.

$$(Z_j | Z_i = z_i) \sim (\tilde{\mu}, \tilde{\rho})$$

$$\tilde{\mu} = \rho_{ji} \rho_{ii}^{-1} z_i$$

$$\tilde{\rho} = 1 - \rho_{ji}\rho_{ii}^{-1}\rho_{ij}$$

where $Z_i = (Z_{j-1}, Z_{j-2}, \dots, Z_{j-n})^T$, $\mu = \begin{pmatrix} \mu_i \\ \mu_j \end{pmatrix}$ with size $\begin{pmatrix} n \\ 1 \end{pmatrix}$ and $\rho = \begin{pmatrix} \rho_{ii} & \rho_{ij} \\ \rho_{ji} & \rho_{jj} \end{pmatrix}$ with size $\begin{pmatrix} n \times n & n \times 1 \\ 1 \times n & 1 \times 1 \end{pmatrix}$

4. Inverse operation of step 1: Transform $\{Z_t\}$ to $\{W_t\}$ with the corresponding empirical distribution by $W_t = F^{-1}(\Phi(Z_t))$. In addition, if W_t is required to take the exact value in $\{X_t\}$, then $W_j = \arg \max_{X_i} X_i \leq W_j$.

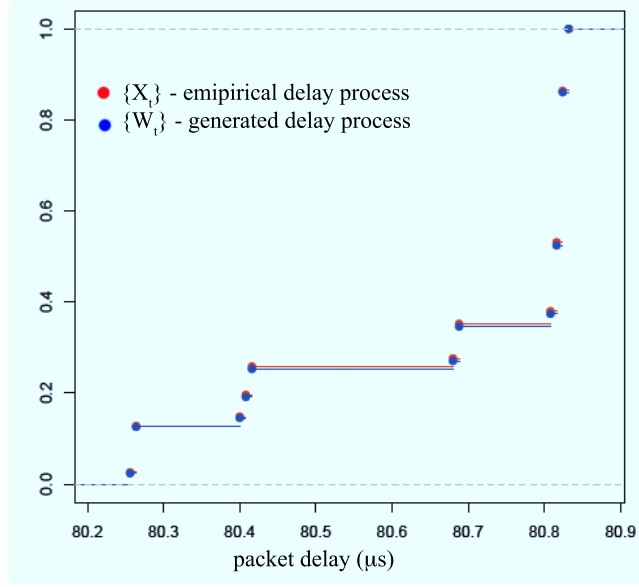


Figure 4.10: CDF of the Delay Processes $\{X_t\}$ and $\{W_t\}$

The above time series generation algorithm is efficient for large-scale simulation, since no manipulations of large matrices are involved. Step 1 and 2 can be done offline to compute ρ , and $\tilde{\rho}$ in step 3 is a constant given ρ . Therefore, the only computations for generating the next delay during runtime are (1) computing $\tilde{\mu}$ in step 3, which is essentially a dot product of two n-dimensional vectors, and (2) one operation to map a Z_t to a W_t in step 4.

Figure 4.10 plots the CDF of $\{X_t\}$ which was 2,000 data points of delay collected from the 3COM switch under high traffic load and the CDF of generated $\{W_t\}$ with 2,000,000 data points. The marginal distribution of both processes are very close. Table 4.4 (see page 33) shows the autocorrelation of

$\{X_t\}$ and $\{W_t\}$. The difference between the corresponding autocorrelations is small and both decrease with increasing n .

CHAPTER 5

MODEL DESIGN AND IMPLEMENTATION

5.1 Switch

The design of the Ethernet switch model in RINSE is shown in Figure 5.1. The multiple interfaces can be attached to a switch. Each interface includes an Ethernet MAC layer and an Ethernet physical layer. During the initialization stage, a MAC address is assigned to each interface and a forwarding table is precomputed for each switch based on the entire network topology specified in the DML configuration file and then loaded into every switch. Using static communication paths can greatly save the simulation running time by shifting path computation offline. The design does not affect the accuracy much since the wired line network topology is pretty stable. Above the layer containing all the interfaces, the switch model has two more layers to take charge of forwarding and processing respectively. The forwarding layer simply reads the destination MAC address from the Ethernet header and looks up the static forwarding table to select the corresponding output port and then push the frame to that particular port. The scheduling layer is the place where all the ideas discussed in Chapter 4 were implemented. The current switch model supports FCFS and WRR scheduling policies. In addition, the switch model is an independent module regardless the underlying protocols, such as switched Ethernet, standard Ethernet or the simple MAC. In other words, frames can travel through a chain of switches connecting by different LAN technologies.

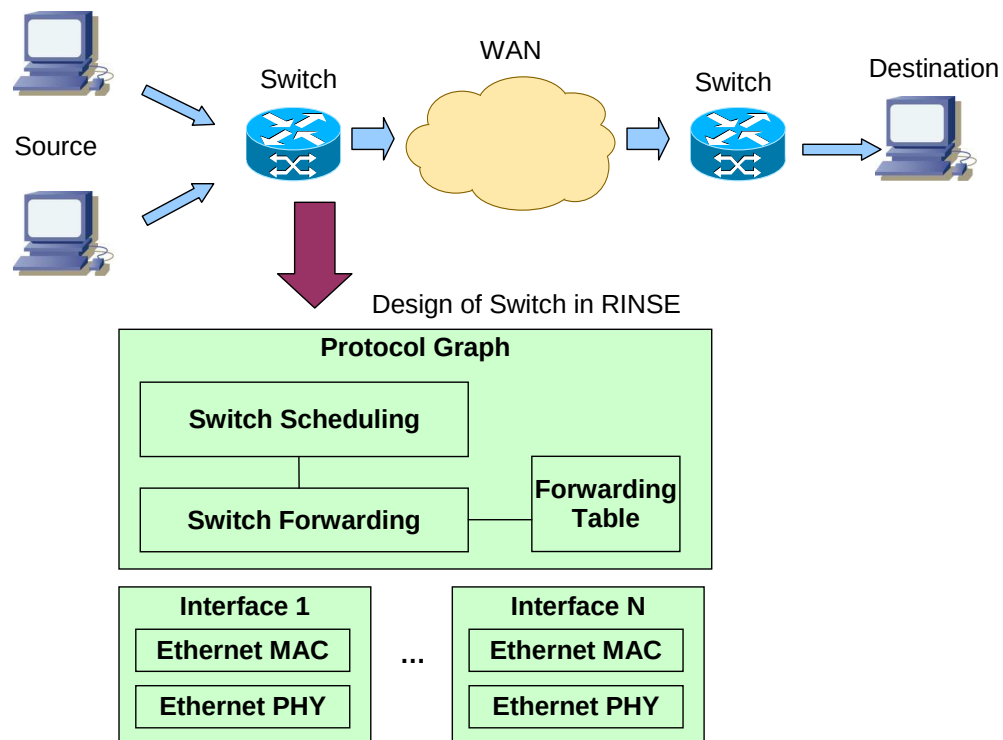


Figure 5.1: Switch Architecture in RINSE

5.2 Ethernet

The simple MAC and physical layers in RINSE are abstract protocol models. They simply receive IP packets, compute the transmission time and propagation delay and then schedule a timer. The callback function explicitly calls every connected host to receive the frame after the scheduled transmission time passes. The destination host then pops up the frames to the IP layer and the rest of the receivers simply discard the unintended frame. We implemented the IEEE 802.3 Ethernet protocol to replace the existing simple MAC and physical layer with protocols that support both CSMA/CD technology for the standard Ethernet and dedicated bandwidth allocation for the switched Ethernet.

Figure 5.2 is an overview of the standard Ethernet model in RINSE. An Ethernet MAC protocol session, an Ethernet PHY protocol session, an Ethernet message model and an Ethernet link model have been developed in the RINSE simulation framework. The Ethernet message model is used for the switched Ethernet protocol as well.

All hosts are attached to the same link, which is the Ethernet cable. The link has the knowledge of the distance between any pairs of hosts, which are specified in the DML file. Only one host can transmit a frame onto the cable at one time; otherwise collision occurs. The CSMA/CD has been implemented as the shared medium access protocol in Ethernet. The Ethernet message handles all the operations related to the Ethernet frame header.

Assumptions on the single Ethernet standard LAN model include: The RTT time is $51.2 \mu\text{s}$, and the maximum Ethernet cable length is 500 m, the number of hosts allowed in one Ethernet LAN is less than 1024.

Ethernet Message

Figure 5.2 shows the format of an Ethernet frame in the model. A preamble has 7 bytes with pattern 10101010 followed by one byte with pattern 10101011. It is used to synchronize receiver and sender clock rates. The next fields are the 6-byte destination MAC address and 6-byte source MAC address. If the Ethernet adapter (implemented as an interface object) receives a frame, whose destination is of its own, or the broadcast address, it strips the Ethernet header and passes the data payload to the IP layer; oth-

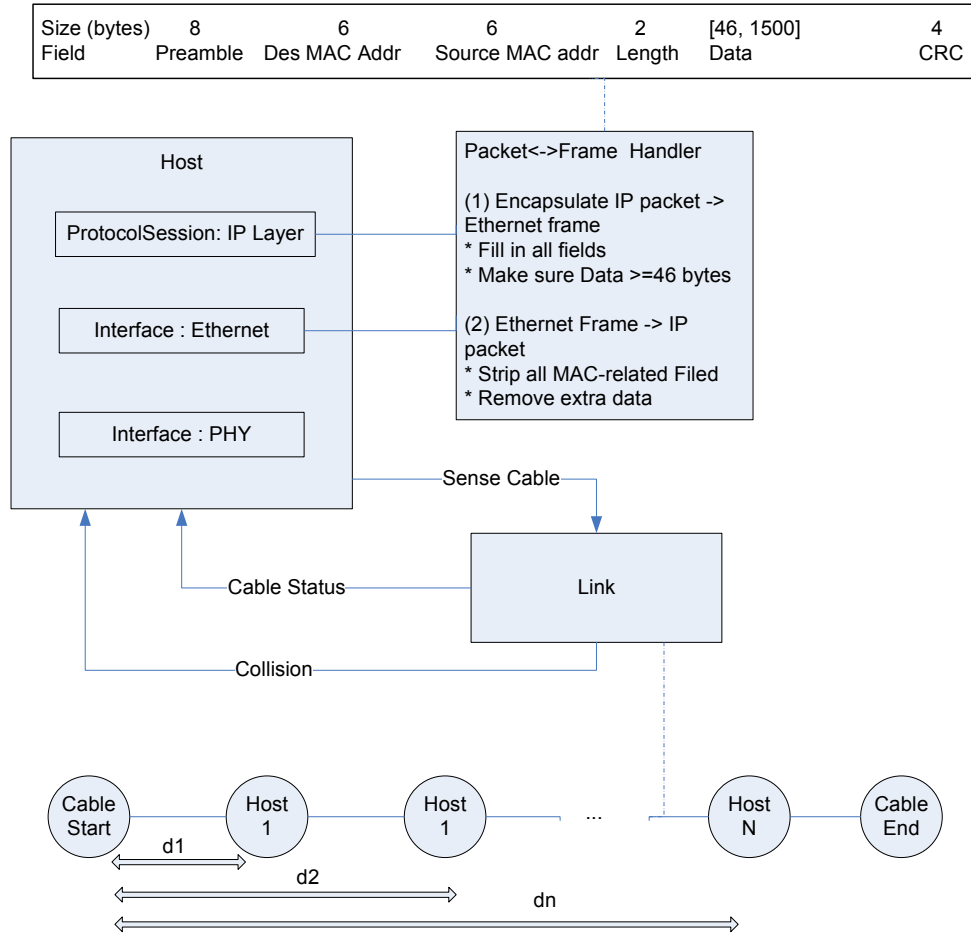


Figure 5.2: Overview of the Standard Ethernet Model in RINSE

erwise the adapter discards the frame. In IEEE 802.3 Ethernet, the 2 bytes after source MAC address represent the frame length; however, in Digital-Intel-Xerox Ethernet Standard, the 2-byte means type. The 4-byte CRC is appended at the end for detecting transmission errors. However, the real CRC functionality is not necessarily required in this model for simulating Ethernet packet transmission and collision detection. Therefore, actual CRC generating and checking functionalities were not implemented for simulation speed gain.

A sender detects collision by comparing what it is currently transmitting with what it is receiving on the wire. Once simultaneous transmission occurs, the two copies are different and collision is detected. The frame size must be large enough so that the transmission time is longer than the maximum RTT

Table 5.1: Dynamic Table in the Link Object

Sender	Starting Time	Ending Time	Collision Time	List of Colliding Nodes
Interface	VirtualTime	VirtualTime	VirtualTime	Interface Vector
0[0]	0	1	NA	NA
1[0]	1	4	3	1[0], 2[0]
2[0]	1	5	3	1[0], 2[0]
1[0]	6	9	NA	NA

time. Therefore, the minimum size is 46 bytes. If the IP packet pushed from upper layer is smaller than 46 bytes, the model will add paddings, simply all zeros, to fulfill the minimum frame size requirement.

Ethernet MAC and Physical Protocol Session

The Ethernet MAC and physical protocol sessions were built with reference to the simple MAC and physical protocol sessions respectively. Both two protocol sessions have four states: Idle, sensingLink, transmittingData, receivingData and collision.

A timer is associated with the Ethernet MAC protocol session. The timer is scheduled to re-sensing cable or handle collision. A MultiFIFO queue is also used in the MAC protocol to hold the packets from the IP layer. The MAC layer only senses the cable status and transmits the first element in a droptail queue. If the queue is full, the incoming IP packet is simply dropped.

Ethernet Link

The Ethernet link represents a single Ethernet LAN. It is responsible for collision detection and answers hosts' queries about the current link status. The link keeps track of all the hosts who have started sending frames and stores the information into a dynamic table as shown in Table 5.1.

An entry is added into the table when a host senses the link is idle and is ready to transmit a frame. An entry is removed from the table when the transmission is successfully completed or a collision occurs if two entries have overlap in transmission time.

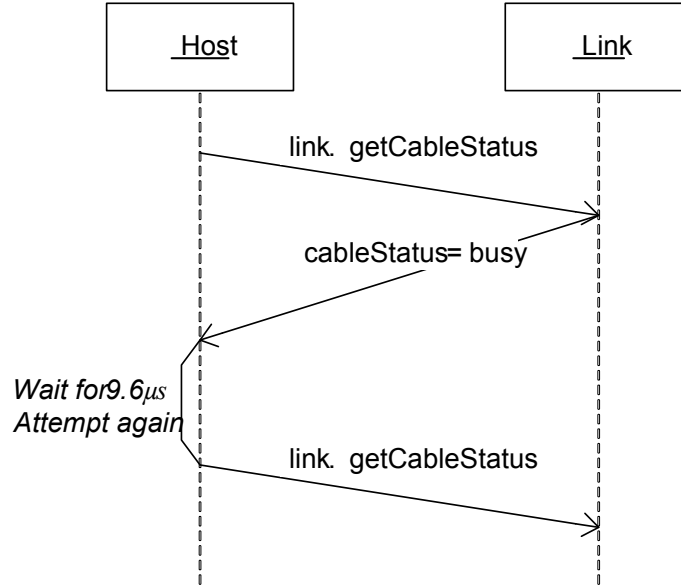


Figure 5.3: Use Case Diagram for Busy Link

The link also knows the distance between any host and start of the cable, which is specified in the DML file, for computing propagation delay. An example of the link attributes in DML file is shown below:

Link [attach 0(0) distance 10 attach 1(0) distance 40 delay 25.6 length 500], where delay = 25.6 μ s (RTT = 51.2 μ s), length = 500 m

When a query on the link status arrives, the link object computes the latency based on the distance between the host who initiates the query and all other hosts listed in the table. If there is at least a sender that starts longer ago than that latency, the host knows that the link status is busy; otherwise, the link status is idle.

CSMA/CD

The design of the CSMA/CD can be classified into the following three cases:

Busy Link A host checks the status of the link before transmitting a frame. If the status is busy, the host has to keep sensing the link until the link gets idle. The actual implementation involves control message passing between MAC layer and PHY layer. Figure 5.3 is a use case diagram for the busy link case.

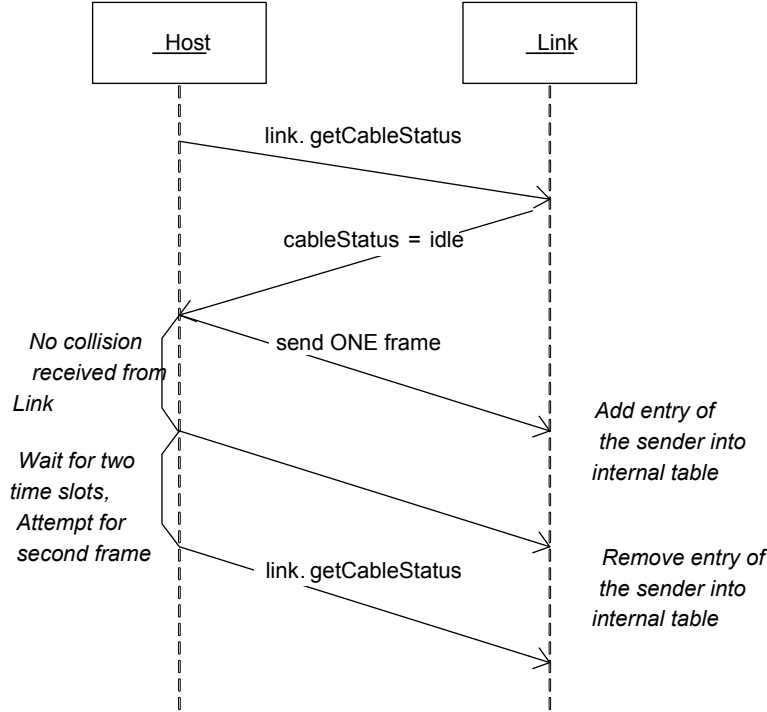


Figure 5.4: Use Case Diagram for Successful Frame Transmission

Successful Frame Transmission Once the link status is idle, a host can send a frame, and a record is placed inside the link object until the frame is completely transmitted. The sender has to wait for another $9.6 \mu\text{s}$ before the next attempt, which tries to maximize the fairness among all hosts. Figure 5.4 shows a use case diagram for the successful frame transmission case.

Collision When a collision occurs, exponential backoff is used to make the retransmitting waiting time more likely to be different among each sender in order to avoid another collision. When the n^{th} collision occurs, the host randomly gets an integer k from $[0, 2^{n-1}]$, and waits $k \cdot RTT$ ($51.2 \mu\text{s}$) before making another attempt. Figure 5.5 shows a use case diagram for the collision case.

We compared the performance of the CSMA/CD standard Ethernet protocol with the simple MAC and physical layer protocol in RINSE. We focus on two metrics: throughput and wall-clock time with respect to offered load, with the following definition.

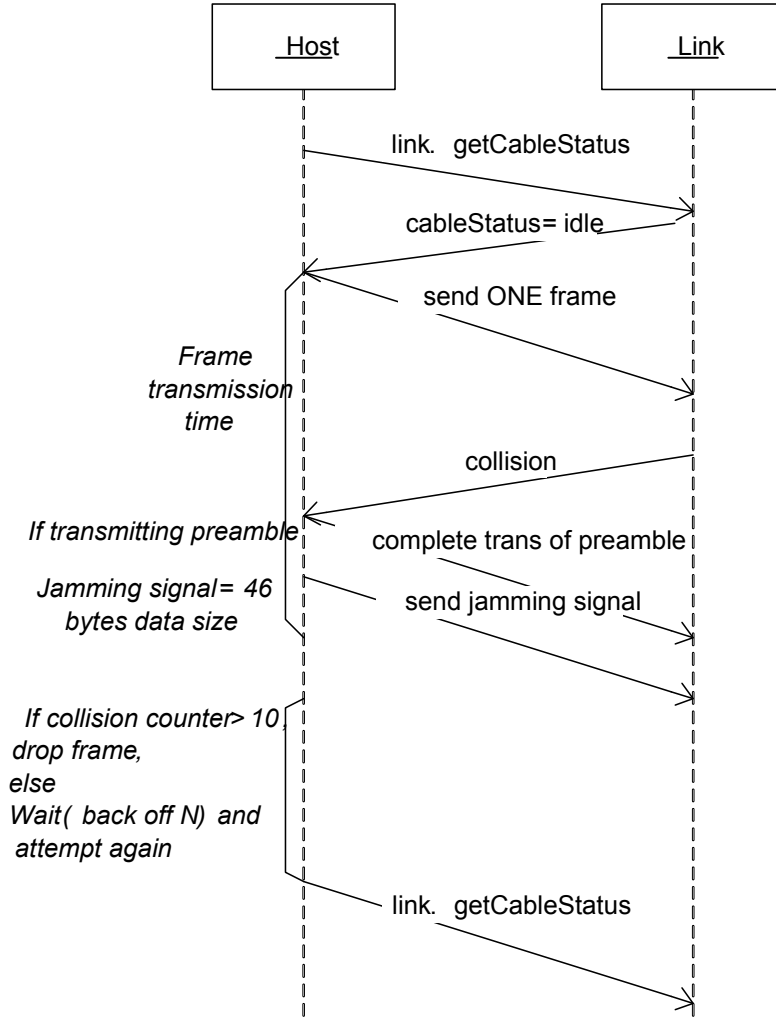


Figure 5.5: Use Case Diagram for Frame Collision

$$R = \frac{n \times S}{T_s}$$

$$Q = \frac{m \times S}{T_{off} + \frac{S}{B}}$$

R	Total throughput at all clients
Q	Offered load rate
n	Number of files successfully downloaded
S	File size
T_s	Total simulation time
T_{off}	Average delay between two successive file download requests, exponentially distributed
B	Link bandwidth
m	Number of client-server pair

The link bandwidth B is fixed to 10 Mb/s and the link delay is fixed to $25.6 \mu\text{s}$, which is half of the RTT for 10 Mb/s 802.3 Ethernet. The backoff time slot is $51.2 \mu\text{s}$ and link length is 500 m. The inter-frame gap is set to $9.6 \mu\text{s}$. The maximum number of backoff before dropping frames is 10.

We had 10 fixed client-server pairs in the network topology. Each client is continuously downloading fixed-size files (100KBytes) from a server through UDP or TCP. The start time of every client was uniformly distributed from 0 to 2 s. The session expired time is exponentially decreasing with an initial value 64 s. The inter-session time at server is zero. We varied the offered load from 1 Mb/s to 20 Mb/s with 1 Mb/s increment by changing T_{off} . The simulation was running for 1000 s. Figure 5.6 and 5.7 compare the throughput and the wall-clock time with all the hosts running UDP. Figure 5.8 and 5.9 show the results with all hosts running TCP connections. We can see that:

- The throughput of Simple MAC unrealistically exceeds the bandwidth (10 Mb/s) for both UDP and TCP test cases.
- The throughput of Simple MAC increases linearly, while the throughput of Ethernet increases fast before the point where the offered load rate is equal to the bandwidth, and increases slowly after that point due to the exponential backoff mechanism.
- Wall-clock time in Ethernet is 5 to 10 times slower than Simple MAC for UDP test case, and 3 to 6 times slower for TCP test case. The extra computation is introduced by (1) CSMA/CD with exponential backoff in Ethernet protocol, such as updating sender information in

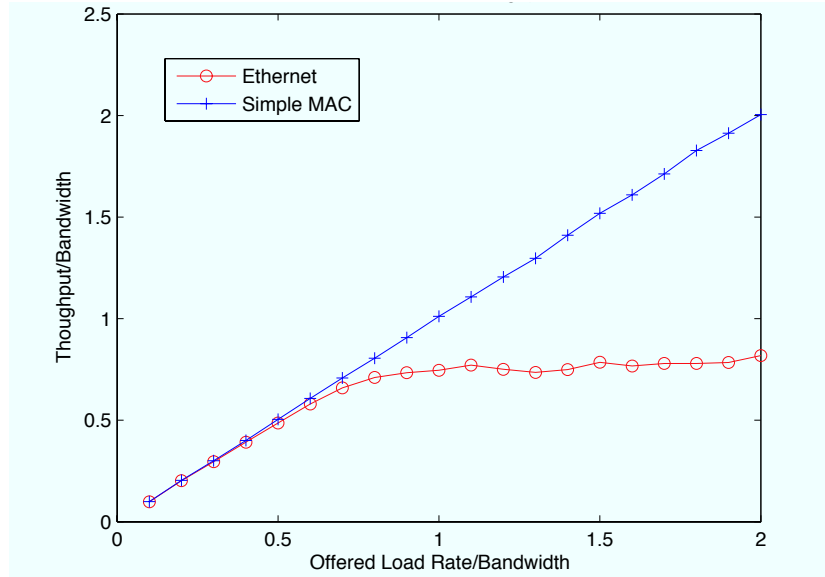


Figure 5.6: Total Client Throughput - UDP

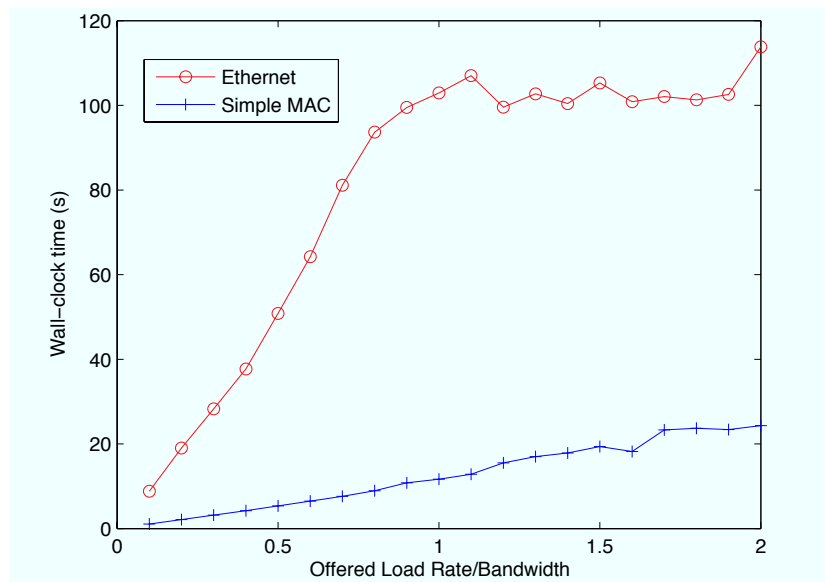


Figure 5.7: Wall Clock Time - UDP

the dynamic table in the link object, timer scheduling and callback functions, (2) extra processing for the Ethernet headers.

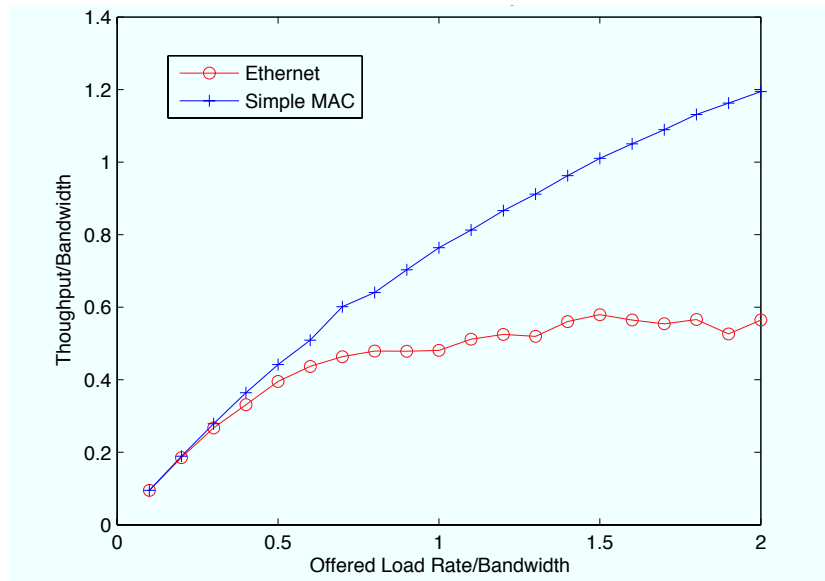


Figure 5.8: Total Client Throughput - TCP

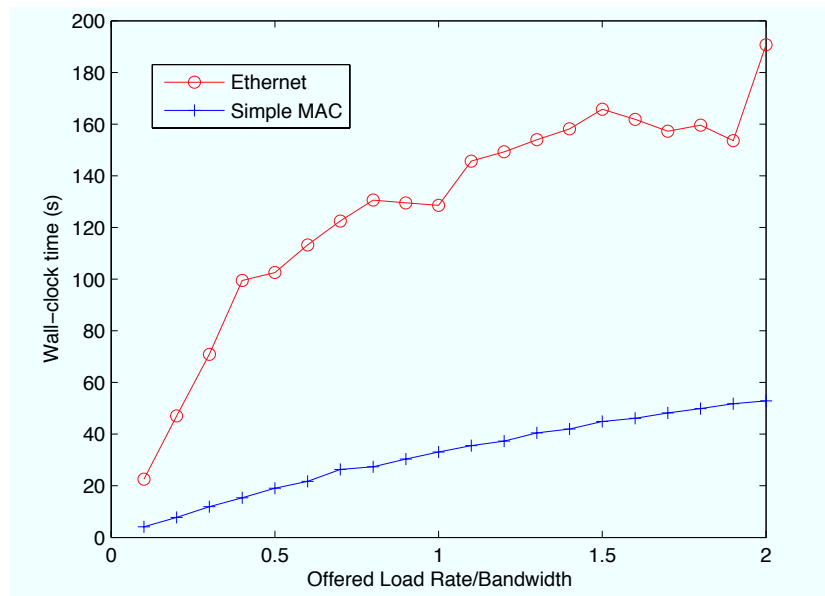


Figure 5.9: Wall Clock Time - TCP

CHAPTER 6

EVALUATION

6.1 Simulation Speed

We have implemented all models discussed in our simulation framework and now evaluate their relative speeds. The framework is built using C++, and includes a significant amount of infrastructure for simulating large-scale wire-line networks. Our speed evaluations are based on a topology that chains a number of switches, illustrated in Figure 6.1. For each switch, two constant bit rate flows were injected using different input ports, and directed to the same output port. The first switch in the sequence takes two flows with bit rates 900 Mb/s and 300 Mb/s. The remainder take one flow from the previous switch, and also another 300 Mb/s flow. In every flow 1 Gbytes were sent. The wall-clock time and number of events were recorded for comparison.

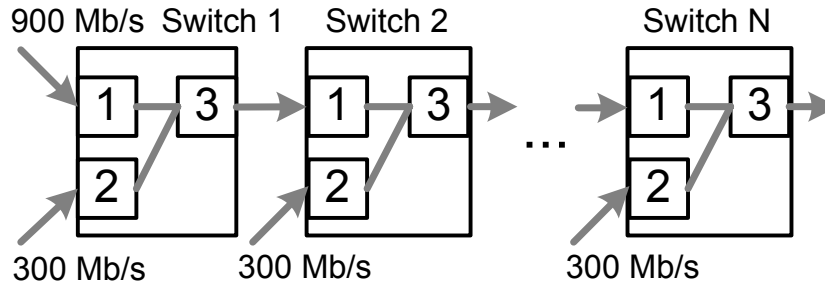


Figure 6.1: Architecture for Simulation Speed Tests

There is a certain overhead due to traffic generation that is amortized as we increase the number of switches. We get the best sense of relative overheads of switch models by increasing the number of switches in sequence.

From the point of view of events, the detailed NetGear implementation has already one event per frame, and already eschews the overhead of queuing.

Table 6.1: Performance of Q1, Q2 and Q3 Switch Models for N=20 Topology

Model	Events	Exc. Time	Time/Event
Q1	12.7 M	32.15 s	2.53 μs
Q2	12.7 M	38.36 s	3.04 μs
Q3	19.9 M	54.04 s	2.7 μs

The real potential for performance improvement of the NetGear latency-approximate model will come later in Chapter 7.

We turn instead to the 3COM switch, where we have implemented and compared the performance of three models. *Q3* denotes the detailed queueing model, in the generalizable module where each frame has a departure event, and an arrival event. *Q2* denotes the module that at the completion of a scheduling round schedules the arrival of frames at their downstream switches, and thus avoids the cost of additional events. Finally, *Q1* denotes the latency-approximate model. Table 6.1 gives raw performance figures for the $N = 20$ topology, and Figure 6.2 shows relative performance figures obtained by varying N . The experiments were executed on a PC with 2.8GHz dual core processor and 2GB RAM. Looking at the raw performance, we see events are relatively lightweight. The larger average event cost of *Q2* over *Q3* is due to the fact that half the events in *Q3* are departure events which have very little work associated with them. There are after all almost twice as many events under *Q3* as there are under *Q2*. Looking at the relative performance we note that by increasing the numbers of switches we increase the relative proportion of simulation workload carried by the network simulation, and see that it pretty well dominates the simulation by the time we are simulating 15 or 20 switches under these loads. The latency-approximate models take only 60% of the time that the full queueing models require, while the optimized latency-accurate version of the 3COM switch takes a little over 75% of that time. The difference in performance between *Q1* and *Q2* is due to *Q1*'s lack of explicit queue management code. We see that our objectives in decreasing the execution requirements of a switch have been accomplished. Perhaps with some cleverness we could speed *Q1* and *Q2* further by piggy-backing invocation of scheduler logic entirely onto arrival events. However,

the lowest ratio of execution time we can hope for is 50%, so the remaining possible performance benefits are considerably less than what we have already achieved.

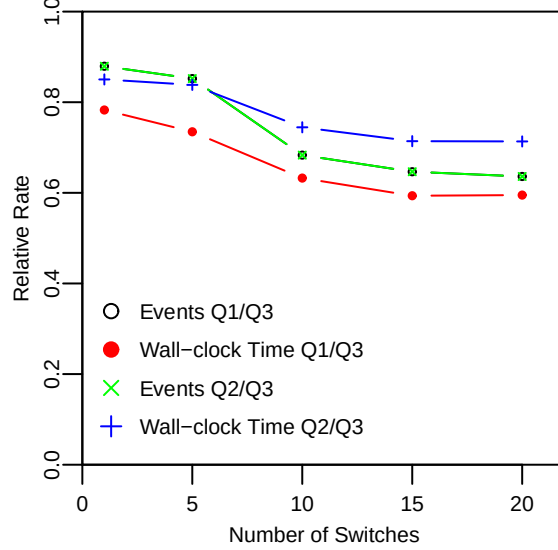


Figure 6.2: 3COM 3CGSU08: Performance

6.2 Frame Loss Modeling Accuracy

We may think of loss behavior of a flow in both the NetGear and 3COM switches in terms of cyclic alternating periods; in one period all frame arrivals are buffered. In the next period all frame arrivals are dropped, and so on. We characterize the statistical behavior of loss in the data, and in different models by analyzing three metrics: loss rate (average number of frames lost per accept/loss cycle), the loss episode (average number of frames lost in burst in the loss state), and the time between loss episodes (i.e., time during which frames are accepted). We perform the evaluation on one switch like that in Figure 3.2 (see page 13), with the same configuration of inputs (with no background flow). Ten million frames were generated for each flow. The sending rate of flow 1 was fixed at 900 Mb/s, while the sending rate of flow 2 was varied with each experiment, ranging from 100 Mb/s to 1 Gb/s, with a 100 Mb/s increment. These rates ensure frame loss in each configuration, and can be compared with real traces run with the same input rates.

Figure 6.3 presents the results for both switches, plotting the mean and standard deviation of each metric. For the NetGear switch, the mean value of all the three loss metrics generated by both models perfectly match the real data. For the 3COM switch, the loss rates perfectly match real data. The real data shows some variation in loss episode at the higher load levels (when both flows experience loss) that the models do not track. The raw magnitude of the difference is not large—1 frame essentially—this data serves to re-emphasize the point that we do not know what is inside the 3COM switch, but have managed a fairly good representation of it.

The results show that the simplified switch models are as accurate as the detailed queueing models in terms of the overall and long-term loss metrics. However, the simplified models do not capture the transient behavior of delay as accurately as do the detailed models as shown in Figure 4.4(a3) and (b3). Still the agreement is quite good, and we accept the inaccuracy as a fair price paid for significant speedup.

There are of course limitations to the experiments and models we present. Real traffic is bursty, and the hardware traffic generator we use generates frames at a constant rate. We are looking into mechanisms for creating more irregularly shaped traffic with the NetFPGA card. The introduction of burstiness will almost surely affect the accuracy of latency and loss our models can achieve. We need still to understand how the switches behave with more severe cross-traffic than we have created to date. We have also to discover whether the scheduling performed within the 3COM switch is applied per logical *flow*, or per *input-queue to output port* pair.

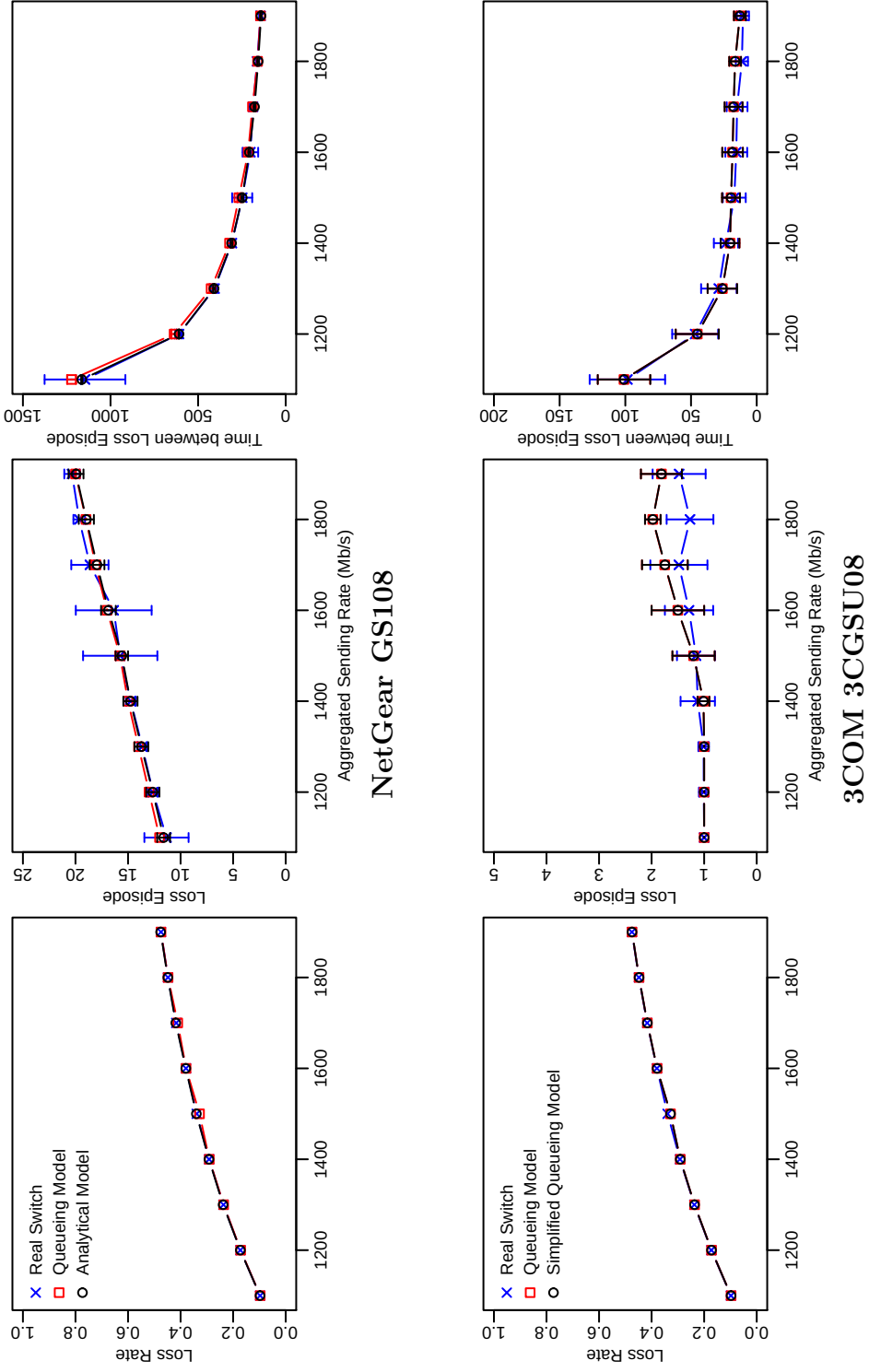


Figure 6.3: Frame Loss Accuracy Evaluation

CHAPTER 7

CONCLUSION AND FUTURE WORK

The work we report is unique in creating a testbed where we can generate Gigabit Ethernet traffic at line rates, and measure precisely what the latency and loss patterns are through a switch, for sequences of millions of frames. We took the measurements from commercial switches, found two very distinct behaviors, and proposed queuing models to represent the switches. For both switches we created one model that faithfully determines both frame loss and latency under its modeling assumptions, and another model that faithfully determines frame loss but uses an estimate for latency. Our goal in creating the simpler model is to reduce the execution cost of handling a frame to almost one event per frame per switch. We validated all models against the real observed traffic, and reduced execution time to 60% of its original value.

Future work lies in reducing the cost of switched network simulation further. Our goal is to approach the simulation of detailed foreground traffic using an approach similar to that developed by Nicol and Yan [44]. Rather than separate the simulation of a frame’s passage across a network with event(s) at each switch, we aim to take advantage of congestion-free subpaths and accelerate the passage of a frame between switches where congestion creates loss and some care is needed in selecting which frames from which flows are lost. The key to such an approach is finding ways of computing sufficiently accurate latencies without detailed simulation, and finding ways of faithfully capturing loss patterns at congested switches. The present thesis shows how to accelerate the usual approach to network simulation, but also aims at this important piece of future work.

REFERENCES

- [1] A. Demers, S. Keshav, and S. Shenker, “Analysis and simulation of a fair queueing algorithm,” in *Symposium Proceedings on Communications Architectures & Protocols*, 1989, pp. 1–12.
- [2] D. Stiliadis and A. Varma, “Design and analysis of frame-based fair queueing: A new traffic scheduling algorithm for packet-switched networks,” *SIGMETRICS Performance Evaluation Review*, vol. 24, no. 1, pp. 104–115, 1996.
- [3] M. Shreedhar and G. Varghese, “Efficient fair queueing using deficit round-robin,” *IEEE/ACM Transactions on Networking (TON)*, vol. 4, no. 3, p. 385, 1996.
- [4] L. Zhang, “VirtualClock: A new traffic control algorithm for packet-switched networks,” *ACM Transactions on Computer Systems (TOCS)*, vol. 9, no. 2, p. 124, 1991.
- [5] D. Ferrari and D. Verma, “A scheme for real-time channel establishment in wide-area networks,” *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 3, pp. 368–379, 1990.
- [6] T. Szigeti and C. Hattingh, “Quality of service design overview,” 2004. [Online]. Available: <http://www.ciscopress.com/articles/article.asp>
- [7] OPNET, “Network modeling and simulation environment,” 2008. [Online]. Available: <http://www.opnet.com/>
- [8] OMNeT, 2008. [Online]. Available: <http://www.omnetpp.org>
- [9] ns 2, “The network simulator,” 2008. [Online]. Available: <http://www.isi.edu/nsnam/ns/>
- [10] “Deter network security testbed,” 2008. [Online]. Available: <http://www.deterlab.net/>
- [11] R. Chertov, S. Fahmy, and N. Shroff, “Emulation versus simulation: A case study of TCP-targeted denial of service attacks,” in *TRIDENT-COM*, 2006, p. 10.

- [12] N. Hohn, D. Veitch, K. Papagiannaki, and C. Diot, "Bridging router performance and queuing theory," in *SIGMETRICS '04/Performance '04: Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, 2004, pp. 355–366.
- [13] R. Chertov, S. Fahmy, and N. Shroff, "A device-independent router model," in *IEEE INFOCOM 2008: The 27th Conference on Computer Communications*, April 2008, pp. 1642–1650.
- [14] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM Computer Communication Review*, vol. 25, no. 1, p. 187, 1995.
- [15] J. Bolot, "Characterizing end-to-end packet delay and loss in the internet," *Journal of High Speed Networks*, vol. 2, no. 3, pp. 305–323, 1993.
- [16] H. Sanneck, G. Carle, and R. Koodli, "Framework model for packet loss metrics based on loss runlengths," in *Proceedings of International Society for Optical Engineering*, vol. 3969, 2000, pp. 177–187.
- [17] M. Yajnik, S. Moon, J. Kurose, and D. Towsley, "Measurement and modelling of the temporal dependence in packet loss," in *IEEE INFOCOM '99: Proceedings of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 1, 1999.
- [18] M. Borella, D. Swider, S. Uludag, and G. Brewster, "Internet packet loss: Measurement and implications for end-to-endQoS," in *Proceedings of the 1998 ICPP Workshops on Architectural and OS Support for Multimedia Applications/Flexible Communication Systems/Wireless Networks and Mobile Computing*, 1998, pp. 3–12.
- [19] A. Feldmann, A. Gilbert, and W. Willinger, "Data networks as cascades: Investigating the multifractal nature of Internet WAN traffic," *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 4, pp. 42–55, 1998.
- [20] A. Feldmann, A. Gilbert, P. Huang, and W. Willinger, "Dynamics of IP traffic: A study of the role of variability and the impact of control," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 1999, p. 313.
- [21] W. Jiang and H. Schulzrinne, "Modeling of packet loss and delay and their effect on real-time multimedia service quality," in *PROCEEDINGS OF NOSSDAV '2000*, 2000.
- [22] S. Moon, J. Kurose, P. Skelly, and D. Towsley, "Correlation of packet delay and loss in the Internet," University of Massachusetts, Amherst, MA, 01003 USA, Tech. Rep., 1998.

- [23] E. Gilbert et al., “Capacity of a burst-noise channel,” *Bell System Technical Journal*, vol. 39, no. 9, pp. 1253–1265, 1960.
- [24] J. Bolot, S. Fosse-Parisis, and D. Towsley, “Adaptive FEC-based error control for Internet telephony,” in *Proceedings of IEEE INFOCOM’99: Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, 1999, pp. 1453–1460.
- [25] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, August 2000.
- [26] L. Peterson and B. Davie, *Computer Networks: A Systems Approach*. Amsterdam: Morgan Kaufmann Publications, 2007.
- [27] M. Liljenstam, J. Liu, D. Nicol, Y. Yuan, G. Yan, and C. Grier, “Rinse: The real-time immersive network simulation environment for network security exercises,” in *PADS ’05: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, 2005, pp. 119–128.
- [28] J. Liu, S. Mann, N. Van Vorst, and K. Hellman, “An open and scalable emulation infrastructure for large-scale real-time network simulations,” in *IEEE INFOCOM 2007: 26th IEEE International Conference on Computer Communications*, 2007, pp. 2476–2480.
- [29] SSF, “Scalable simulation framework,” 1999. [Online]. Available: <http://www.ssfnet.org/>
- [30] “Domain Modeling Language (DML) reference,” 1999. [Online]. Available: <http://www.ssfnet.org/SSFdocs/dmlReference.html>
- [31] H. Weibel and D. Bechaz, “Implementation and performance of time stamping techniques,” in *Proceedings of the 2004 Conference on IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, vol. NISTIR, no. 7192, 2004.
- [32] G. Covington, G. Gibb, J. Lockwood, and N. McKeown, “A packet generator on the netfpga platform,” in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2009.
- [33] G. Watson, N. McKeown, and M. Casado, “NetFPGA: A tool for network research and education,” in *Workshop on Architecture Research using FPGA Platforms*, 2006.
- [34] C. Satten, “Lossless gigabit remote packet capture with linux,” 2007. [Online]. Available: <http://staff.washington.edu/corey/gulp/>
- [35] “ENDACE DAG network monitoring cards,” 2009. [Online]. Available: <http://www.endace.com>

- [36] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, “Weighted round-robin cell multiplexing in a general-purpose ATM switch chip,” *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 8, pp. 1265–1279, 1991.
- [37] G. Chuanxiong, “SRR: An $O(1)$ time complexity packet scheduler for flows in multi-service packet networks,” in *SIGCOMM '01: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2001, pp. 211–222.
- [38] M. Cario and B. Nelson, “Autoregressive to anything: Time-series input processes for simulation,” *Operations Research Letters*, vol. 19, no. 2, pp. 51–58, 1996.
- [39] B. Biller and B. L. Nelson, “Modeling and generating multivariate time-series input processes using a vector autoregressive technique,” *ACM Transactions on Modeling and Computer Simulation*, vol. 13, no. 3, pp. 211–237, 2003.
- [40] M. Cario and B. Nelson, “Modeling and generating random vectors with arbitrary marginal distributions and correlation matrix,” IEMS Northwestern University, Evanston, IL, USA, Tech. Rep., 1997.
- [41] B. Melamed, “The empirical TES methodology: Modeling empirical time series,” *Journal of Applied Mathematics and Stochastic Analysis*, vol. 10, no. 4, pp. 333–353, 1997.
- [42] R. Nelsen, *An Introduction to Copulas*. New York, NY: Springer, 2006.
- [43] A. Sklar, “Fonctions de répartition à n dimensions et leurs marges,” *Publications at Institute of Statistics, University of Paris*, vol. 8, pp. 229–231, 1959.
- [44] D. Nicol and G. Yan, “High-performance simulation of low-resolution network flows,” *Simulation*, vol. 82, no. 1, p. 21, 2006.