Efficient Gigabit Ethernet Switch Models for Large-scale Simulation

Dong Jin, David M. Nicol, and Matthew Caesar

University of Illinois at Urbana-Champaign
{dongjin2, dmnicol, caesar}@illinois.edu

Keywords: switch modeling, simulation, measurement, performance

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 paper describes a unique testbed that gathers highly accurate measurements of loss and latency through a switch, 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), while providing excellent accuracy for loss and adequate accuracy for latency.

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 behavior 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. Simulators like OPNET[11] and OMNeT++[10] have detailed switch models that capture complex internal architectures for different types of switches. 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 simula-

978-1-4244-7291-8/10/$26.00

tions where the interest is less on the network and more on the applications and protocols running on the network. Various types of switch models already exist in network simulators and emulators. In ns-2[9] and DETER[1], a simple first-come, first-served (FCFS) queuing model is used for every type of switch. As have others[3], 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 investigated in [4][7]. Accuracy is improved by taking specific data from real devices into consideration. However, it is still hard to adapt one model to all type of switches once the placement of the queues are fixed, and the simulation speed is slower than that of the simple FCFS queue model.

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), Fair Queueing(FQ), Weighted Round Robin(WRR)[12]. 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. 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 switch models that support

- 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,
- straight forward model development for new switches.

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[13].

In this paper 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 patterns. Based on this data, we develop 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.

The remainder of this paper is structured as follows. Section 2 describes the experiment setup. Section 3 analyses the experimental data and presents the models we have developed to explain the data. Section 4 evaluates the simulation speed and model accuracy. Section 5 discusses the future work.

2. Measurement

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.

We built a testbed that uses hardware to instrument, transmit, and capture Ethernet frames at line rates. Figure 1 depicts our solution. Traffic is generated using a 4-port NetFPGA card[6][14]; the frames to send can be loaded from a pcap file, and the sending rate is specified. 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 this case a 4.5G4 Endace DAG card[2]. The Endace card stamps received frames with a clock having a 10-nanosecond resolution; the card can also capture and store millions of frames with zero loss at 1Gb/s. 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 timestamps is the delay through the switch. For example, in Figure 1 a "Traffic 1" frame is sent simultaneously out on ports 1 and 3 of the NetFPGA card. One instance arrives at port 2 of the DAG and is time-stamped on receipt. The other

enters the switch via port 2, is routed out via port 8, and arrives at the DAG on port 1 where it is time-stamped. The difference in time-stamps is the delay through the switch (and time on the wire from switch to DAG). To validate the approach, we replaced the switch with a wire and calculated the difference under a 1Gb/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 as shown in Figure 1, the testbed can monitor two flows in real time. Both cards are placed on the same PC (four dual-core 2.0GHz CPUs running CentOS 5.2). We captured data from three 8-port Gigabit Ethernet switches: 3COM 3CGSU08, NetGear GS108v2 and TrendNet TEGS80G. They are all simple low-end commodity 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 in pcap format, and a sequence number was added 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 timestamps 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 to achieve maximum sending/receiving rate by minimizing the affect 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 behavior of the TrendNet and NetGear switches were entirely similar in all cases. Correspondingly, the rest of the paper will only show results from the NetGear and 3COM switches.

3. Analysis and Modeling

3.1. Data Analysis

The first set of experiments monitored a single flow whose sending rate varies from 100Mb/s to 1Gb/s with 100Mb/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 2 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 1Gb/s and the variance is close to 0μs. 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.

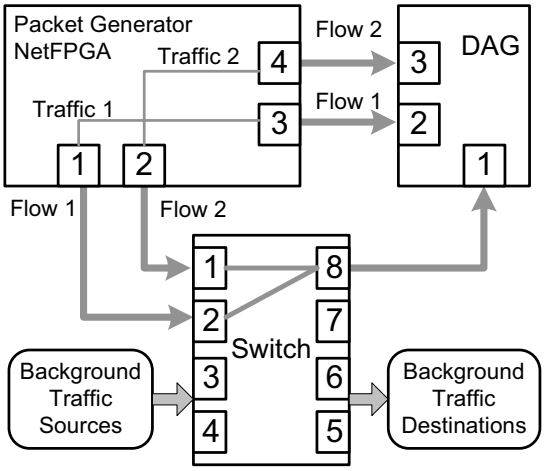

Figure 1. Testbed Setup

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 1Gb/s CBR flows and five 1Gb/s CBR flows from five input ports to one output port (see again Figure 1), *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 1Gb/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 2 as the baseline non-queuing delay added by a switch.

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

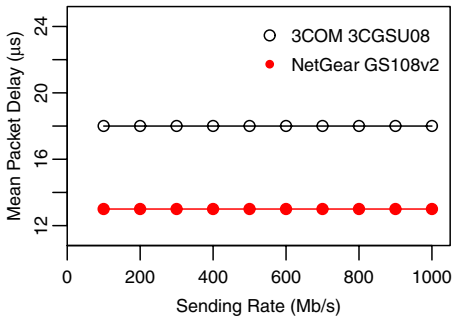

Figure 2. Frame Delay for Single Traffic Flow

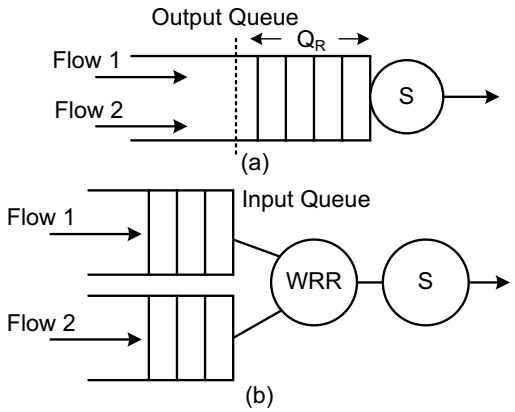

Figure 3. (a) Output Queue Model for NetGear GS108v2 (b) Input Queue Model with WRR for 3COM 3CGSU08

the two switches have significantly different delay and loss patterns. Figure 5(a1) and (b1) show one sample pattern of two input flows with sending rates 900Mb/s and 300Mb/s—5(a1) describes the 3COM switch and 5(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 900Mb/s and 300Mb/s flows, in bursts, while the 3COM switch dropped frames only from the 900Mb/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 500Mb/s (not shown here). 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. 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.

3.2. Queueing Model

The loss and delay pattern observed in the NetGear switch can be well explained by FCFS output queue as shown in Figure 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 3(b)) where frames are in queues associated with their input ports (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* [12]. 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 don't 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[8]. Actually, many commercial switches are using round robin based scheduling due to the low time complexity O(1) and low implementation cost[5]. 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 ex-

ample, if $\lfloor Q_2/Q1 \rfloor = N$, then N frames from queue 2 are served while one frame from queue 1 is served.

With a 900Mb/s sending rate in flow 1 and 300Mb/s in flow 2, the ratio of queue lengths should be approximately 3:1 in steady state, and the effective service rates are 700Mb/s and 300Mb/s respectively. Therefore, for the 300Mb/s flow, no loss was observed and the stabilized delay should be smaller than that of the 900Mb/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 500Mb/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 987Mb/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 5(a2) and (b2) plot the delays corresponding to the same real switch test case as shown in Figure 5(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 however 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 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 however.

3.3. Latency-Approximate Models

In our drive to achieve high performance switch simulations we will at some point have to knowingly tradeoff 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 however 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.

3.3.1. 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.

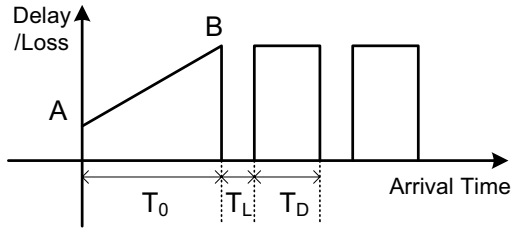

Figure 4. Simple Model

The pattern may be parametrized by the variables below:

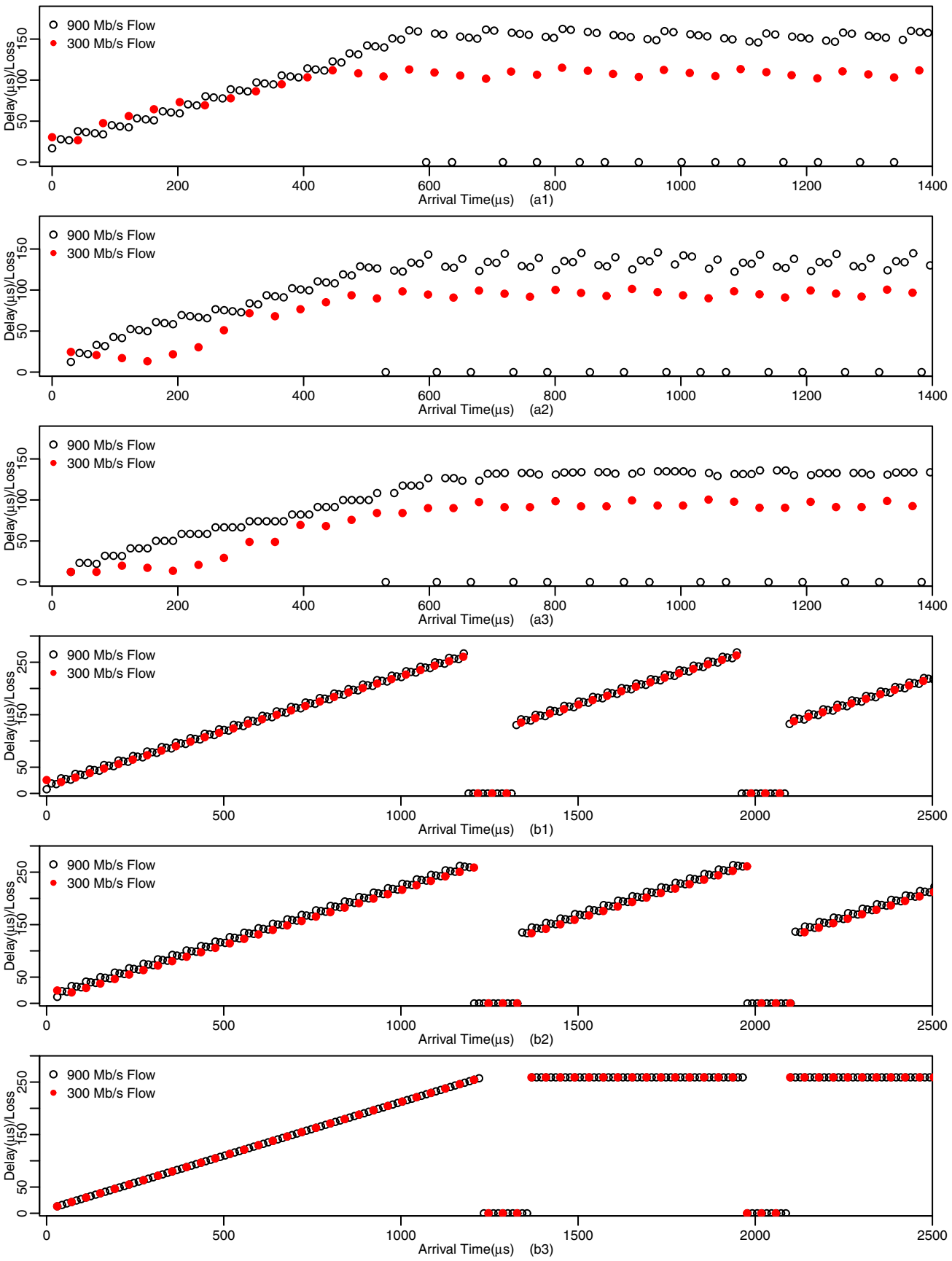

Figure 5. Delay/Loss Pattern, Two Flows to One Output Port, (a1) 3COM Real (a2) 3COM Queueing Model (a3) 3COM Simplified Queueing Model (b1) NetGear Real (b2) NetGear Queueing Model (b3) NetGear Analytical Model

A	Minimum latency, (see Figure 2)
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 5(b3) show 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 5(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.

3.3.2. 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 benefit in parallel 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 ar-

rives 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 deque 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 $j = 1, 2, \ldots N_j$, let $L_{k,j}$ be the delay of k^{th} of these. At each scheduling round,

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

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 5(a3) shows the simulation results using the 3COM latency-approximate model with α=0.9 for the same test case in Figure 5(a2). The simplified model generates frame loss patterns as accurate as those of the detailed model, with no losses in the low load flow and 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. Evaluation

4.1. 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. 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 900Mb/s and 300Mb/s. The remainder take one flow from the previous switch, and also another 300Mb/s flow. 1Gbytes were sent in every flow. The wall-clock time and number of events were recorded for comparison.

There is a certain overhead due to traffic generation that is amortized as we increase the number of switches. We get

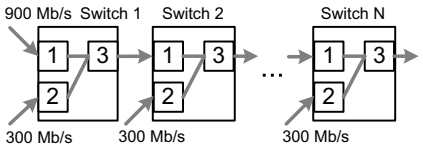

Figure 6. Architecture for Simulation Speed Tests

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. The real potential for performance improvement of the NetGear latency-approximate model will come later (in ways described as "Future Work".

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. Figure 7 gives raw performance figures for the $N = 20$ topology, and 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 proportional 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 queuing 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.

Model	Events	Exc. Time	Time/Event
Q1	12.7M	32.15s	2.53μs
Q2	12.7M	38.36s	3.04μs
Q3	19.9M	54.04s	2.70μs

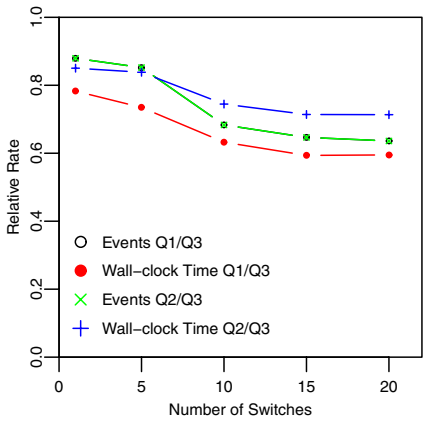

Figure 7. 3COM 3CGSU08: Performance

4.2. 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 1, 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 900Mb/s, while the sending rate of flow 2 was varied with each experiment, ranging from 100Mb/s to 1Gb/s, with a 100Mb/s increment. These rates ensure frame loss in each configuration, and can be compared with real traces run with the same input rates.

Figure 8 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 don't 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 5(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.

5. 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 [15]. 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 to finding ways of faithfully capturing loss patterns at congested switches. The present pa-

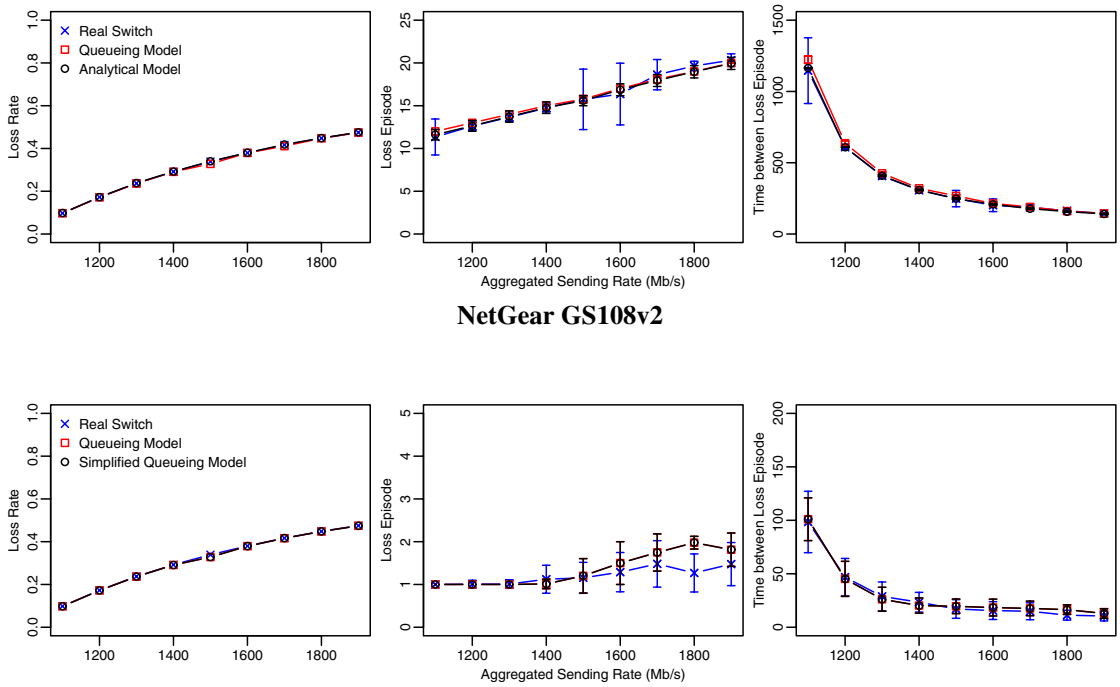

Figure 8. Frame Loss Accuracy Evaluation

per shows how to accelerate the usual approach to network simulation, but also aims at this important piece of future work.

6. Acknowledgement

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 authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Deter network security testbed. *http://www.deterlab.net/*.

[2] ENDACE DAG network monitoring cards. *http://www.endace.com*.

[3] R. Chertov, S. Fahmy, and N. Shroff. Emulation versus simulation: A case study of TCP-targeted denial of service attacks. In *TRIDENTCOM*, page 10, 2006.

[4] R. Chertov, S. Fahmy, and N. Shroff. A device-independent router model. *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 1642–1650, April 2008.

[5] 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*, pages 211–222, New York, NY, USA, 2001. ACM.

[6] 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.

[7] 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*, pages 355–366, New York, NY, USA, 2004. ACM.

[8] 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*, 9(8):1265–1279, 1991.

[9] ns 2. The network simulator. *http://www.isi.edu/nsnam/ns/*.

[10] OMNeT. *http://www.omnetpp.org*.

[11] OPNET. Network modeling and simulation environment. *http://www.opnet.com/*.

[12] D. Stiliadis and A. Varma. Design and analysis of frame-based fair queueing: a new traffic scheduling algorithm for packet-switched networks. *SIGMETRICS Perform. Eval. Rev.*, 24(1):104–115, 1996.

[13] T. Szigeti and C. Hattingh. Quality of service design overview, 2004.

[14] G. Watson, N. McKeown, and M. Casado. NetFPGA: A tool for network research and education. In *Workshop on Architecture Research using FPGA Platforms*, 2006.

[15] D. N. G. Yan. High performance simulation of low-resolution network flows. *Simulation : Transactions of the Society for Modeling and Simulation International*, 82(1):21–42, Jan. 2006.