Proceedings of the 2010 Winter Simulation Conference
B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.

FAST SIMULATION OF BACKGROUND TRAFFIC THROUGH FAIR QUEUEING NETWORKS

Dong Jin
David M. Nicol

Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Urbana, IL, 61801, USA

ABSTRACT

Large-scale network simulation is widely used to facilitate development, testing and validation of new and existing network technologies. To ensure a high-fidelity experimental environment, we often need to embed real devices and have the simulator running faster than real time. Since the generation and movement of background traffic in a network simulation represents so much of the workload, we develop here techniques for modeling background traffic through switches that use Fair Queueing scheduling. Our work is an extension of earlier efforts that assumed all switches use First-Come-First-Serve scheduling. It turns out the the scheduling policy has an important impact on the logic of the earlier technique, and on the performance it delivers. We describe the algorithm and give experimental results that show that like the earlier work, very significant acceleration of background traffic simulation is achieved.

1 INTRODUCTION

Network simulation is widely used to study protocols and applications running on large-scale networks because of its low cost and flexibility. However, the cost of simulating the network can easily overwhelm the overall cost of performing the simulation experiment. In some applications, only a small fraction of traffic is of specific interest. For example, in models of the power grid we may be more interested in the detailed behavior of some specific flows (e.g. a DNP3 or Modbus connection between a control station and a particular substation), and be interested in other flows only in so far as they consume resources and affect the behavior of the detailed flows. In experiments where real devices are embedded into the simulator, it is challenging to meet the real-time constraint given the fact that the network being simulated has hundreds of thousands of devices and (typically) tens of thousands of flows represented at any given time. Therefore, there is a strong motivation to have efficient multi-resolution traffic models, in which background traffic is modeled with much less details than foreground traffic.

A high performance background traffic model is proposed and validated in (Nicol and Yan 2006). The model aggregates the background flows between network access points, transforms the dependencies among the flows into a reduced system of non-linear equations, and efficiently solves the system using fixed-point iteration. One foundational assumption of the model is that the network switches and routers manage queueing using First-Come-First-Serve (FCFS) scheduling. However, in reality, such devices often have schedulers that provide rate proportional service (Stiliadis and Varma 1996), such as round robin (RR), weighted round robin (WRR) (Katevenis et al. 1991) and virtual clock (Zhang 1991). Indeed, many commercial switches use round robin based scheduling due to the low time complexity O(1) and low implementation cost (Guo 2001).

In this paper, we investigate two Gigabit Ethernet switches, 3COM 3CGSU08 and NetGear GS108v2 based on the traces collected with a unique testbed. The testbed we created can generate and capture 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. The experimental results reveal the FCFS behavior in the NetGear switch and Fair Queuing

(FQ) behavior in 3COM. The testbed setup, experimental results and analysis are reported in Section 2. Based on analysis of the real data, we developed an algorithm for the FQ scheduler, integrated it with the existing algorithm based on the FCFS scheduler. The new version of the background traffic model is presented in Section 3. We then validated the model in terms of convergence behavior, simulation speed and accuracy, and compared the simulation results between the FCFS networks and the FQ networks in Section 4.

2 MEASUREMENT

2.1 Testbed

A comprehensive study of a Gigabit Ethernet switch, such as the frame loss and delay patterns, output rate and drop rate of each flow, 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(a) depicts our solution. Traffic is generated using a 4-port NetFPGA card (Covington et al. 2009) (Watson et al. 2006); the frames to send can be loaded from a pcap file with user-specified sending rate. A 4.5G4 Endace DAG card (Endace 2009) is the receiving end, which places time-stamps on received frames using a clock that has 10 ns resolution; the card can also capture and store millions of frames with zero loss at 1 Gb/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. Two identical flows are generated from the NetFPGA to the same destination, for example, in Figure 1(a) a frame is sent simultaneously out on ports N1 and N4 of the NetFPGA card. One instance arrives at port D4 of the DAG and is time-stamped on receipt. The other enters the switch via port S1, is routed out via port S5, and arrives at the DAG on port D1 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 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. This design does not require clock synchronization between the NetFGPA card and the DAG card.

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 frame size is fixed at 1500 bytes to minimize the inter-frame gap and so maximize the sending/receiving rate. We generated one million frames for every flow, in every experiment. Post analysis of received frames can identify the missing frames by analyzing the sequence numbers. 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. In addition, we can also calculate the output rate and the drop rate of each flow. Both cards are placed on the same PC (four dual-core 2.0GHz CPUs running CentOS 5.2). We captured data from two 8-port Gigabit Ethernet switches: 3COM 3CGSU08 and NetGear GS108v2. They are all simple low-end commodity switches with no configuration interface available, and every port is identical. All ports were connected by cat-6 Ethernet cables.

2.2 Experiment and Data Analysis

The first set of experiments (see Figure 1(a)) monitored a single flow whose sending rate varies from 100 Mb/s to 1 Gb/s with 100 Mb/s increment, with no background traffic. We observed that the delay in both switches behaves constantly with no loss, about 13 μs for the NetGear switch and 18 μs for the 3COM switch. The delay added by each switch has the same mean value under any sending rate up to 1Gb/s and the variance is close to 0 μs. This shows that the switches handle line rate without loss, the switches have significantly different delays, and that with deterministic inter-arrival times those delays are constant.

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 (see Figure 1(b)) and five 1 Gb/s CBR

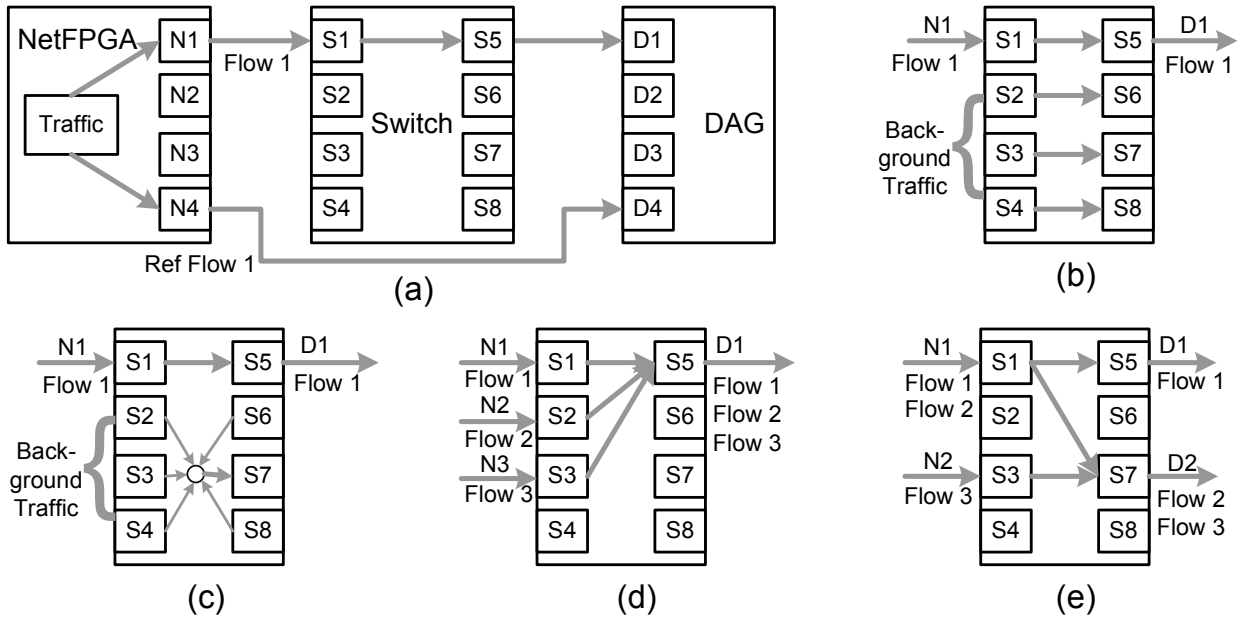


Figure 1: Testbed Overview and Experiment Setup

flows from five input ports to one output port (see Figure 1(c)), *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 (1) both switches can handle nearly 1 Gb/s flows at each output —no frame losses were observed until the total rate of the flows to the same output port reached 988 Mb/s, and (2) flows coming from different source ports and going to different destination ports do not affect each other.

The third set of experiments monitored the output rate of three flows from three input ports to the same output port, as shown in Figure 1(d). Flow 1 moves at 100 Mb/s, flow 2 moves at 500 Mb/s, and flow 3 varies from 100 Mb/s to 1 Gb/s with a 100 Mb/s increment. Table 1 shows the results from the NetGear switch and Table 2 shows the results from the 3COM switch. When total input rate of the three injected flows does not exceed the switch's service capacity (which is 1 Gb/s in theory and 988 Mb/s as observed), both switches experienced zero drop rate on every flow. When the total input rate exceeds the maximum service rate, the NetGear switch dropped frames in all the three flows and the drop rate is almost same. However, for the 3COM switch, we observed that (1) all frames in flow 1 (100 Mb/s) were received, and (2) frames in flow 2 and 3 ($\geq$ 500 Mb/s) equally share the remaining bandwidth. The implication is that all flows have the same bandwidth reservation ($\approx$ 333 Mb/s) and the switch served the connections in proportion to their reservation, and then distributes the extra bandwidth equally among the active flows.

To investigate switch behavior, we collected the individual flow's detailed delay and loss patterns. By utilizing all four ports of the NetFPGA card, the testbed can monitor delay and loss patterns of two flows. We removed flow 3 in Figure (d) and ran the fourth set of experiments to monitor flow 1 and flow 2. The sending rates on both flows were varied from 100 Mb/s to 1Gb/s with a 100 Mb/s increment. Figure 2(a) and (b) show one sample pattern of two input flows with sending rates 900 Mb/s and 300 Mb/s—2(a) describes the 3COM switch and 2(b) 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. 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 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 distinctive behavior of the two switches can be explained by different scheduling policies as studied in (Jin et al. 2010). The scheduler in the NetGear switch and the 3COM switch can be modeled respectively by a FCFS server and a weighed round robin server, which belongs to the class

Table 1: NetGear, Results of the Three Input Flows to One Output Port Experiment

Input Rate (Mb/s)			Output Rate (Mb/s)			Drop Rate		
Flow 1	Flow 2	Flow 3	Flow 1	Flow 2	Flow 3	Flow 1	Flow 2	Flow 3
100	499	100	100	499	100	0	0	0
100	499	200	100	499	200	0	0	0
100	499	299	100	499	299	0	0	0
100	499	399	98	492	394	0.02	0.01	0.01
100	499	499	90	447	447	0.10	0.10	0.10
100	499	599	83	414	487	0.17	0.17	0.19
100	499	699	77	383	525	0.23	0.23	0.25
100	499	799	72	359	554	0.28	0.28	0.31
100	499	899	68	339	578	0.32	0.32	0.36
100	499	988	65	316	604	0.35	0.37	0.39

Table 2: 3COM, Results of the Three Input Flows to One Output Port Experiment

Input Rate (Mb/s)			Output Rate (Mb/s)			Drop Rate		
Flow 1	Flow 2	Flow 3	Flow 1	Flow 2	Flow 3	Flow 1	Flow 2	Flow 3
100	499	100	100	499	100	0	0	0
100	499	200	100	499	200	0	0	0
100	499	299	100	499	299	0	0	0
100	499	399	100	486	399	0	0.03	0
100	499	499	100	443	442	0	0.11	0.11
100	499	599	100	443	442	0	0.11	0.26
100	499	699	100	443	442	0	0.11	0.37
100	499	799	100	443	442	0	0.11	0.45
100	499	899	100	443	442	0	0.11	0.51
100	499	988	100	442	443	0	0.11	0.55

of rate proportional server (Stiliadis and Varma 1996).

In the fifth set of experiments, we injected three flows into the switch, where flow 1 and flow 2 shared the same input port (S1) and flow 2 and flow 3 shared the same output port (S7), as shown in Figure 1(e). The input rate of flow 3 is fixed to the line rate (988 Mb/s), which ensures the sum of flow 2 and flow 3 would always be greater than the maximum service rate and would result frame losses at outport S7. We want to observe flow 2's impact on flow 1 when flow 2 itself is congested. Table 3 shows the output rates for both switches. The results show that the output rate of flow 1 is always equal to its input rate for both switches. Further experiments actually reveal the same behavior of delay and loss on flow 1 as we saw in the first experiment. Therefore, flow 2 does not affect flow 1 at all. This can be explained by the virtual output queue, in which frames are classified according to their destination MAC addresses upon arrival at the input port. The frames with different destination MAC will then be processed independently. In addition, flow 2 and flow 3 share the bandwidth identically as in the third and fourth experiments. We learn then that flows going to different destination ports do not affect each other. It is reasonable to simplify a switch model by modeling its output ports individually and independently. Therefore, our proposed background traffic model uses the output port as a basic element. Details will be discussed in next section.

We modified the input pcap file of flow 1 by redirecting the source IP address on half of the frame to a different source IP address, and used the modified traffic input to conduct all the five sets of experiments again. The results were exactly same, which means the two switches use only source and destination MAC address to differentiate

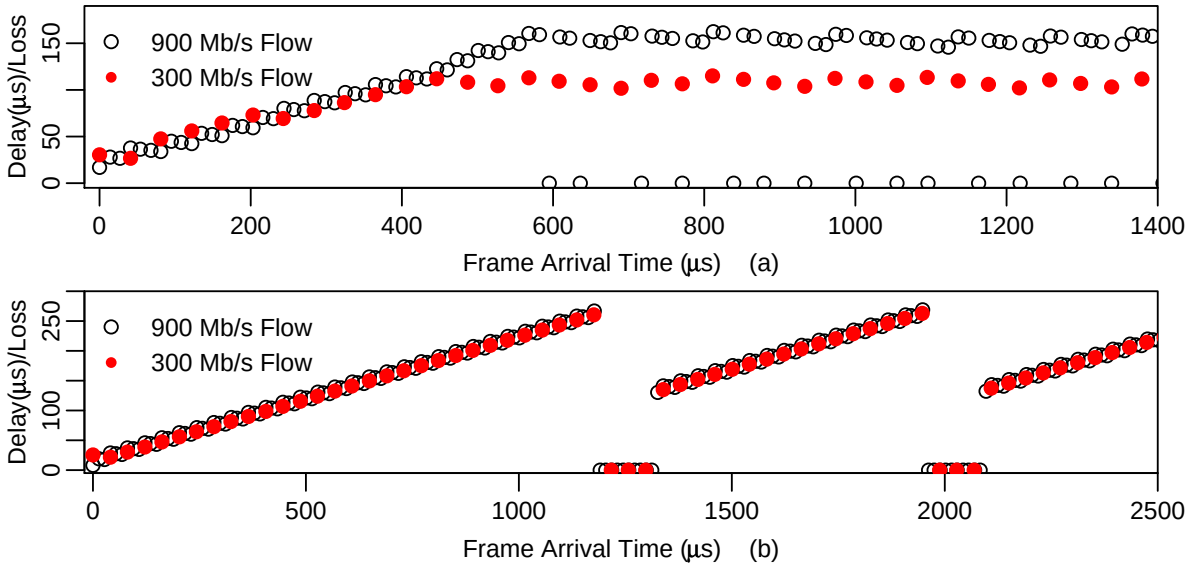


Figure 2: Delay/Loss Pattern, Two Input Flows to One Output Port, (a) 3COM (b) NetGear

Table 3: Experimental Results: Three Flows with Two Sharing One Input Port and Two Sharing One Output Port

Input Rate (Mb/s)			NetGear Output Rate (Mb/s)			3COM Output Rate (Mb/s)		
Flow 1	Flow 2	Flow 3	Flow 1	Flow 2	Flow 3	Flow 1	Flow 2	Flow 3
99	889	988	99	466	521	99	493	495
198	790	988	197	435	552	197	491	497
329	659	988	329	404	583	329	492	496
494	494	988	493	338	649	493	492	496
659	329	988	658	258	729	658	329	658
790	198	988	790	167	820	790	197	790
889	99	988	888	90	897	888	99	888

flows, nothing above the layer 2 header is used by the switches. Therefore, it is reasonable to aggregate flows on a per input port basis to output port pair.

From all the experiments, we learned the following important points which served as guidelines for our background traffic modeling.

- Different switches have different scheduling policy, FCFS (as revealed in the NetGear switch) and FQ (as revealed in the 3COM switch) are two common types.
- Flows going to different output ports do not influence each other, and thus a switch model can be divided into several indepedent output port models.
- Flows with the same input and output port can be aggregated.
- If the sum of input rates is no greater than the maximum service rate, i.e. the line rate, each flow's output rate is equal to it's input rate with zero loss.

3 COARSE-GRAINED BACKGROUND TRAFFIC MODELING AND SIMULATION

3.1 Problem Formulation

The network being studied consists of n access points; the backbone is built using Gigabit Ethernet switches. We are interested in how the network shapes coarse-grained background traffic flows along the communication path from an ingress node to an egress node. In our model, the simulation time is discretized into units of length Δ, and we use t_k to denote $k \cdot \Delta (k = 0, 1, 2...)$. For any ingress-egress pair $< P_i, P_j > (1 \leq i, j \leq n)$, we use $T_{i,j}(t)$ to denote the ingress rate of the corresponding aggregate flow at time t. Since the simulation time is advanced by constant time-step Δ, we need to discretize the ingress rate for every aggregated flow. During the time-step $[t_k, t_{k+1}]$, the traffic volume injected at ingress node i to egress node j is $\int_{t_k}^{t_{k+1}} T_{i,j}(t)\ dt$. The burstiness at time scales smaller than Δ is smoothed. To ensure that the same amount of traffic is generated, the ingress rate at the discretized time $k \cdot \Delta$ is $\frac{1}{\Delta} \times \int_{t_k}^{t_{k+1}} T_{i,j}(t)\ dt$.

All the nodes in our model are connected with unidirectional links, understanding that we can model a bi-directional link as a pair of these. The sending endpoint of a link is attached to a switch's output port. When multiple aggregate flows multiplex at the output port at time t_k, the bandwidth allocation to each flow is governed by the scheduling policy, which varies from switch to switch. A switch is modeled as a group of output ports according to the experimental results in Section 2.2.

We assume that the resource consumption of certain types of flows as well as the forwarding table of a switch change relatively infrequently. We can define a reasonable time-step, say the maximum TCP round-trip time among all flows. This allows the traffic demand generation logic to be responsive to network conditions at a TCP time-scale (i.e., react to loss on a flow). The resource consumption and the data forwarding paths are determined/re-evaluated only at the beginning of the time-step, and treat the consumption and paths as invariant for the rest of the time-step. These flows can be made sensitive to those flows modeled with higher resolution, but only at the fixed update points.

The model must capture the competition between foreground and background flows for link bandwidth. At the beginning of each time-step, for each port, we compute an estimated foreground input rate based on recent observations (or even known predictions, if such were available). As we compute new flow rates, for that port, we treat the estimated foreground flow exactly as any other flow. The fair contribution of foreground traffic is thus considered as we allocate bandwidth for the background traffic.

3.2 Model and Algorithm Description

The goal of our algorithm is this: given a description of flow intensities at ingress nodes, efficiently determine link loads throughout the intermediate switches, and determine flow intensities at the egress nodes. Each flow is associated with a flow rate and a state variable, which can be settled, bounded or unsettled. A settled flow has a finalized flow rate; a bounded flow has a known upper bound on its flow rate; a unsettled flow is neither bounded or settled. A switch is modeled as a few independent output ports. Each port is in one of the three states: resolved, transparent and unresolved. A port is resolved if all its input flows are settled. A port is transparent if not all of its input flows are known, but the sum of the input flow rate is less than the maximum service rate. A port is unresolved if it is neither resolved or transparent. The experimental results in Section 2.2 show that for switches using either FCFS scheduling or FQ scheduling, all the output flow rates can be determined at a resolved port; at a transparent port, a flow's output rate is identical to its input flow rate.

The algorithm proposed in (Nicol and Yan 2006) works well on networks whose switches use FCFS scheduling. It transitions through four phases, each time-step: flow update computation, reduced graph generation, fixed-point iterations and residual flow update computation. Phase I propagates the flow rate and state from ingress points throughout the network; the goal is to settle flows and resolve as many ports as possible. We are necessarily left with circular dependencies among some flow variables, the remaining phases address these. Phase II identifies all the flow variables whose values are circularly dependent, and constructs a dependency graph whose vertex set is composed of all output ports with unresolved state and non-zero out degree. The unknown flow rates are the solution of a non-linear set of equations. Phase III sets up and solves these equations using fixed point iteration. Finally, phase IV substitutes the solutions into the system and finishes the computation.

To modify this algorithm for FQ switching we must reformulation some of the phase I rules to be applied at FQ switches. Other phases are the same; details and validation can be found in (Nicol and Yan 2006). The parameters used by the phase I algorithm are explained as follows:

S_p State of port $p \in \{Resolved, Transparent, Unresolved\}$
n_p Total number of input flows of port p
U_p Bandwidth of port p, i.e. the maximum service rate of port p
F_p Set of input flows passing through port p
$S_{i,p}^{in}$ State of input flow i at port $p \in \{Settled, Bounded, Unsettled\}$
$S_{i,p}^{out}$ State of output flow i at port $p \in \{Settled, Bounded, Unsettled\}$
$\lambda_{i,p}^{in}$ Input rate of flow i passing through port p
$\lambda_{i,p}^{out}$ Output rate of flow i passing through port p
$\Lambda_{in,p}$ Sum of all input flow rates into port p
$\Lambda_{in,p}^{settled}$ Sum of all settled input flow rate of port p
$R_{i,p}$ Reserved bandwidth for input flow i of port p, for FQ
E_p Total extra bandwidth available (bandwidth reserved but not actually used) for port p, for FQ
m_p Number of flows not yet processed at port p, for FQ

In the initialization stage of each time-step, all the ingress nodes generate traffic. Thus, the ingress nodes are in the resolved state, and all the corresponding output flows are settled. All the switches and egress nodes are in the unsettled state and all their connected flows are in the unsettled state. The changes at the ingress nodes then propagate through the entire networks through a 3-step algorithm and loops until no more ports and flows change their rates and states.

Step 1 is to decide the state of a port p based on the input flows.

$$S_p = \begin{cases} Resolved & if \quad \forall i \in F_p, S_{i,p}^{in} = Settled \\ Transparent & if \quad \Lambda_{in,p} \leq U_p \\ Unresolved & else \end{cases}$$

Step 2 calculates the rate and state of all the output flows of port p. Rules are selected based on the port state and the scheduling policy of the switch. They are summarized in Table 4. For a resolved port, all the input flow rates are known, therefore all the output flow rates can be computed and all the out flow states can be determined. For a transparent port, the rate and state of all the output flows is same as the corresponding input flows. For a unresolved port, the goal is to determine the upper bound of its output flows. If the corresponding input flow is settled, the upper bound is derived by ignoring all bounded input flows. If the corresponding input flow is bounded, the upper bound is derived by ignoring all other bounded input flows and treat itself as settled with rate set to the bounded rate.

Step 3 is to pass the rate and state of the output flow to the next switch's input along the flow's path.

$$\lambda_{i,p+1}^{in} = \lambda_{i,p}^{out}, S_{i,p+1}^{in} = S_{i,p}^{out}$$

4 EVALUATION

We now turn to an experimental evaluation of our algorithms. The topology built for the experiment has 294 hosts, 882 directional links and 11,760 flows. The link bandwidth is 1 Gb/s. The switches used in the network all use the FCFS schedulers or the FQ schedulers for comparison. We use a Possion-Pareto-Burst-Process (PPBP) (Zukerman et al. 2003) traffic model to generate the ingress rates for each flow. A Poisson process generates arrivals, each of which contributes a random traffic volume that is sampled from a Pareto distribution. The PPBP is recognized as a good model of traffic arrival to a backbone. We adjust its input parameter to achieve desired average link utilization. The Hurst parameter for the Pareto distribution is set to 0.8 for all the experiments.

Our evaluation is focused on execution speed and accuracy, both relative to a pure packet simulation. The experimental design sets a target average link utilization and chooses PPBP parameters that produce such loadings. In all experiments a time-step is 1 second of simulation time, and each experiment advances simulation time 1000 seconds. Each time-step we measure how many ports end up in dependency cycles, and how many fixed point iterations are required to solve the dependencies. For each target link utilization we run 10 experiments. Our performance metrics are based on observations from all time-steps.

Table 4: Rules of Phase I, Flow Update Computation

Port State	Scheduling Policy	Algorithm
Resolved	FCFS	$\forall i \in F_p, S_{i,p}^{out} = Settled, \lambda_{i,p}^{out} = \lambda_{i,p}^{in} \times min(1, \frac{U_p}{\Lambda_{in,p}})$
	FQ	Initially, sort F_p according to $\lambda_{i,p}^{in}$, $E_p = 0, m_p = n_p$; Every input flow processing round, $\lambda_{i,p}^{out} = \begin{cases} \lambda_{i,p}^{in} & if \quad \lambda_{i,p}^{in} \le R_{i,p} \\ min(\lambda_{i,p}^{in}, R_{i,p} + \frac{E_p}{m_p}) & if \quad \lambda_{i,p}^{in} > R_{i,p} \end{cases}$ $m_p = m_p - 1, E_p = E_p + R_{i,p} - \lambda_{i,p}^{out}$
Transparent	FCFS FQ	$\forall i \in F_p, \lambda_{i,p}^{out} = \lambda_{i,p}^{in}$ and $S_{i,p}^{out} = S_{i,p}^{in}$
Unresolved	FCFS	$\lambda_{i,p}^{out} = \begin{cases} \lambda_{i,p}^{in} \times min(1, \frac{U_p}{\Lambda_{in,p}^{settled}}) & if \quad S_{i,p}^{in} = Settled \\ \lambda_{i,p}^{in} \times min(1, \frac{U_p}{\Lambda_{in,p}^{settled}+1}) & if \quad S_{i,p}^{in} = Bounded \end{cases}$ $S_{i,p}^{out} = Bounded$
	FQ	$\forall j \in F_p \wedge S_{j,p}^{in} = Bounded, \quad \lambda_{j,p}^{in} = 0 \quad if \quad S_{i,p}^{in} = Settled$ *Initially*, $\forall j \in F_p \wedge S_{j,p}^{in} = Bounded \wedge j \neq i, \quad \lambda_{j,p}^{in} = 0 \quad if \quad S_{i,p}^{in} = Bounded$ Compute $\lambda_{i,p}^{out}$ using the same algorithm for FQ resolved port case $S_{i,p}^{out} = \begin{cases} S_{i,p}^{in} & if \quad \lambda_{i,p}^{in} \le R_{i,p} \\ Bounded & else \end{cases}$

4.1 Convergence Behavior

We consider a solution at a time-step to be converged when the maximum relative difference between successive approximations to any flow rate is 0.001, for example, when $|\lambda_{n+1} - \lambda_n|/\lambda_n \le 0.001$ for all flow rates λ_n. In addition, the desired average link utilization is 90% in all the convergence study experiments.

Figure 3 describes the distribution the number of ports on the dependence graph generated in phase II (over all time-steps and all runs), and the distribution of the number of iterations used to reach fixed-point solutions in phase III, for the FCFS network and the FQ network respectively. This data shows that even under high network load (90% link utilization) our algorithm significantly reduces the number of ports involved in the fixed point iteration, and finds the solution in a small number of iterations.

The data shows that the FQ structure tends to induce fewer ports in dependency cycles (7.6% for the FCFS network and 2.2% for the FQ network), and (consequently) fewer iterations of the fixed point algorithm for solution. We understand the difference as being due to the fact that the reserved bandwidth for an individual flow in the FQ switch can produce more settled flows. For FCFS switches under congestion all the output flows change if one input flow changes rate, whereas for FQ switches, an input flow within its reserved bandwidth will not change its output rate.

4.2 Simulation Speed

The algorithm must execute quickly if it is to be useful for large scale network simulation. The authors in (Nicol and Yan 2006) examine the speedup of the algorithm for FCFS networks relative pure packet simulation for background

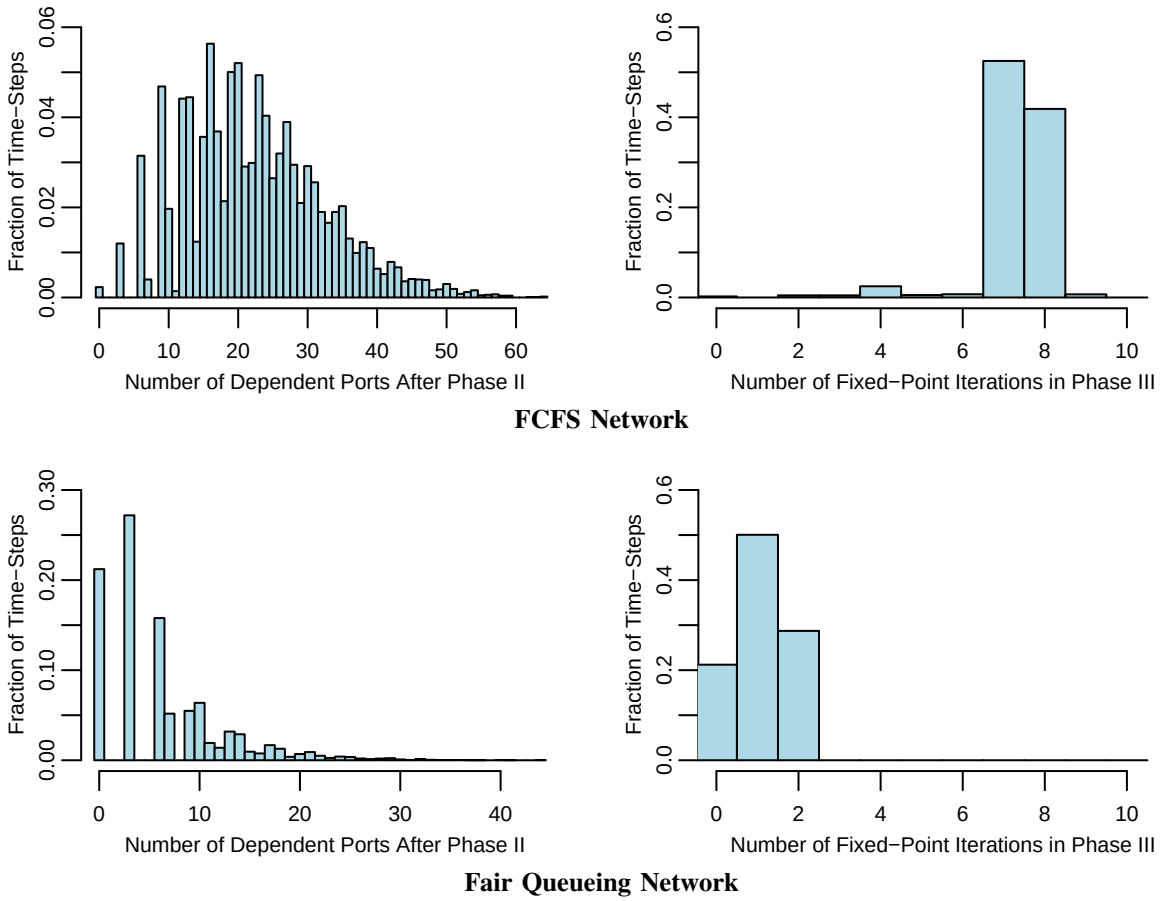


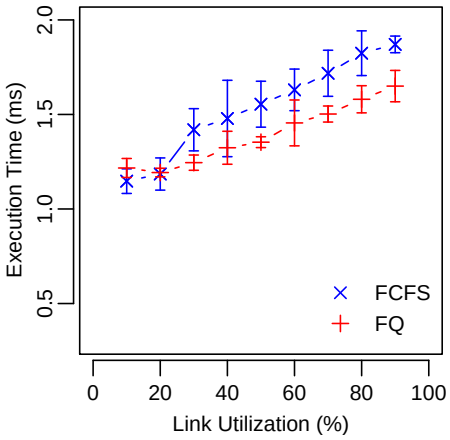
Figure 3: Convergence Experiment Results, 90% Link Utilization

flows and observe speedups exceeding 3000 when executing on an ordinary PC. We infer that similar speedups will be achieved using our algorithm on FQ networks by comparing execution speed with FQ switches with execution speed on FCFS switches, under exactly the same topology and exactly the same traffic loads (varied from 10% to 90%).

These experiments use the same topology as was used in the convergence experiments, and likewise the time-step is 1 simulation second. As before, for each load setting we executed 10 independent runs, each of 1000 simulation seconds. The average execution time per time-step of both switches as well as the corresponding standard derivations are presented in the upper portion of Figure 4 and the average time spent on each phase is also listed under link utilization 10% and 90% for comparison.

Each time-step advances the simulation clock by 1 second, while the total time required under both traffic loads, and both scheduling policies is less than 2 msec. Clearly on this sized model the background traffic calculation in no way threatens the ability to run the simulation faster than real time.

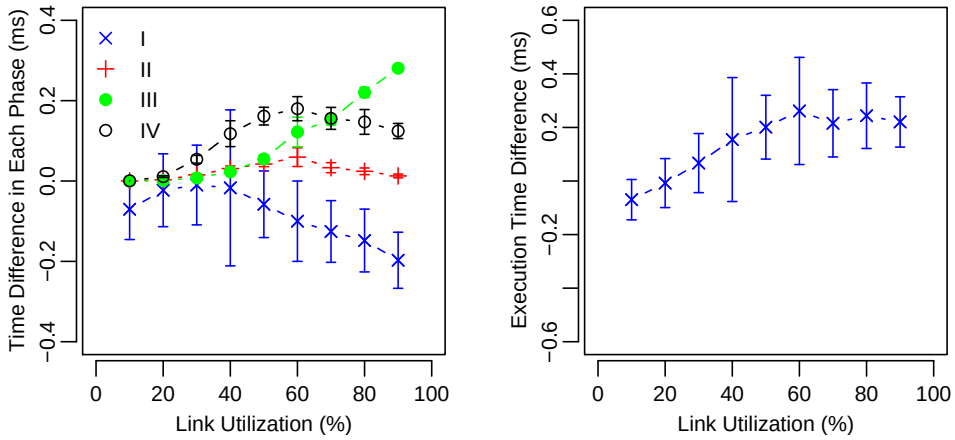
The lower portion of Figure 4 describes the difference in execution time between the topology with FCFS switches, and the topology with FQ switches. For each phase we measure the time required each run. We drive the FCFS model and the FQ model with exactly the same samples of traffic demand using synchronized random number streams, so each data point in the estimate is of coupled sampled paths. Here we see the higher overhead in FQ of phase I due to sorting input flows by rate, and related management. However, for all but the lightest load, the extra computational cost is more than accounted for by accelerated performance in phases 3 and 4, due to fewer unresolved ports, and flow values settling faster because of bandwidth reservations. Under very light load there are very few congested ports and so the additional overhead in FQ switches in phase 1 is not recovered from by improved performance later.



Link Utilization = 10%					
Scheduler	Phase I	Phase II	Phase III	Phase IV	Total
FCFS	1.146	0.00021	0.00014	0.001	1.147
FQ	1.217	0	0	0	1.217

Link Utilization = 90%					
Scheduler	Phase I	Phase II	Phase III	Phase IV	Total
FCFS	1.028	0.086	0.328	0.429	1.871
FQ	1.225	0.074	0.047	0.304	1.650

Execution Time per Time-Step (ms)



Impact of Scheduling Policy on Execution Time

Figure 4: Analysis of Execution Time

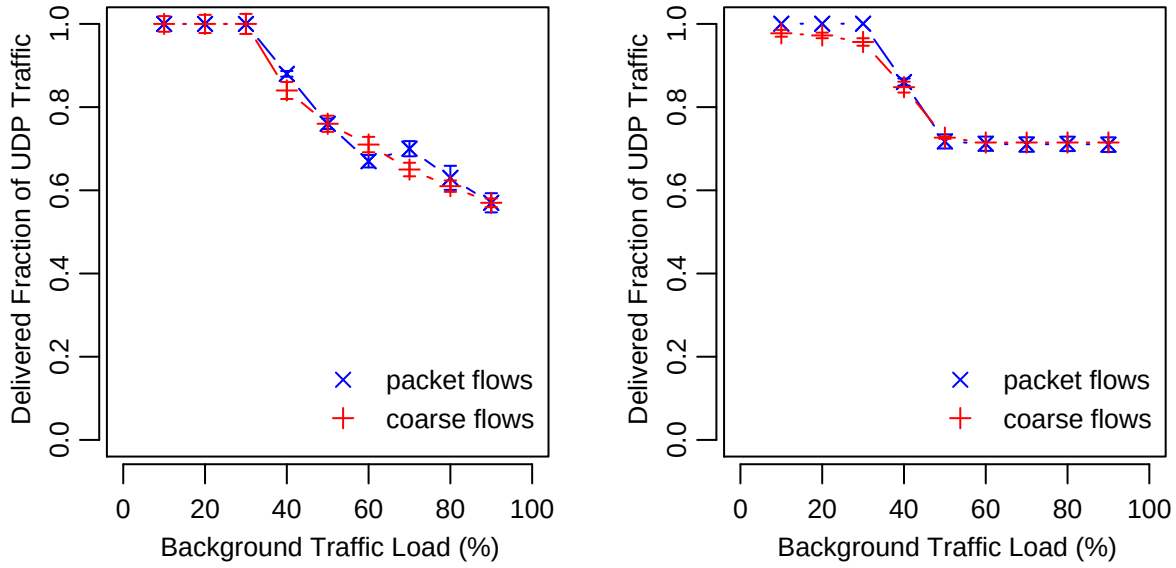
4.3 Accuracy

We also consider the accuracy of our approach when used as a background traffic generator, as compared to the pure packet-oriented approach. The idea is to study the properties of a small number of detailed packet-oriented UDP flows, while the background traffic is either packet based or flow based. Toward this end, we modify the topology used in the convergence simulation experiments by attaching a UDP server to each ingress node and a UDP client to each egress node. The background traffic is simulated with two different techniques: our time-stepped coarse-grained simulation and conventional packet-level simulation. We parametrically control the traffic parameters to cause the average background traffic load on a link to vary between experiments, from 10% to 90%. The PPBP traffic model is used to generate ingress traffic for each flow.

For every background traffic load, we simulate 10 experiments. In each experiment, a UDP client randomly picks a UDP server on any other end host. A UDP server's behavior can be modeled as an ON/OFF process: it uses UDP protocol to send a file of 70 Mbytes at the constant rate of 700 Mb/s, and after it finishes the transfer, it remains idle for an exponentially distributed period with a mean of 1 seconds; after the off period finishes, it sends another 70 Mbytes at the same rate, and the above process repeats until the simulation is over. Every experiment is simulated for 1000 seconds (in simulation time).

We compare the behavior of the UDP flows using different methods for background traffic generation on both the FCFS network and FQ network respectively. The experiments are coupled, in the sense that the same pattern of requests is simulated for the UDP flows in the two experiments that use different background flow generation, and furthermore, the background flow generation in the two experiments is driven by the same offered load. In this way, we ensure that on an experiment-by-experiment basis, we are comparing precisely the same context for measuring foreground UDP flows against different background generation techniques.

Figure 5(a) and 5(b) plot the sample mean and 95% confidence intervals of the delivered fraction of UDP traffic measured in 10 experiments. We see that the fraction of the delivered UDP traffic decreases with the link utilization in both cases due to congestion. The most important feature of the plots is that the mean of the delivered foreground traffic is close between two background flow generation techniques for both FCFS and FQ networks. The FQ network performs better than the FCFS network when the background traffic load is greater than 50%. The explanation here is that we have to approximate packet loss at a congested switches. The more heavily congested the network, the more heavily the approximation is exercised. That approximation is the cause of the error in the FCFS network; in the FQ network, when all the input flows, including the foreground flows, to a output port is higher than its reserved bandwidth, every flow simply gets its own reserved bandwidth, and therefore the packet loss approximation is quite accurate under high traffic load.



(a) FCFS Network

(b) Fair Queuing Network

Figure 5: Delivered Fraction of Foreground UDP Traffic

5 CONCLUSION

This paper extends the early work on simulating network traffic flow at a coarse scale. The early model has a fundamental assumption that the network switches manage queueing using FCFS scheduling, and we expand the model to support fair queueing scheduler, which is widely used in the commercial switches today. The design of the model, such as the way multiple input flows share the bandwidth of an output port, the independent behavior among output ports and aggregation of flows having same input port - output port pair, are validated from the data collected on the real switches using our unique testbed. We also find that the model with fair queueing scheduler results in many fewer dependent ports and thus requires less fixed-point iterations than the model with FCFS scheduler, especially under high load traffic. The new model still runs extremely fast relative to packet-based flow simulation and the impact on foreground flow is still accurate enough for our purposes.

ACKNOWLEDGMENTS

This material is based upon work supported by the Department of Energy under Award Number DE-OE0000097. This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

REFERENCES

Covington, G., G. Gibb, J. Lockwood, and N. McKeown. 2009. A packet generator on the netfpga platform. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*.

Endace 2009. DAG network monitoring cards.

Guo, C. 2001. 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*, 211–222.

Jin, D., D. Nicol, and M. Caesar. 2010. Efficient Gigabit Ethernet switch models for large scale simulation. In *PADS '10: Proceedings of the 24th Workshop on Principles of Advanced and Distributed Simulation*.

Katevenis, M., S. Sidiropoulos, and C. Courcoubetis. 1991. Weighted round-robin cell multiplexing in a general-purpose ATM switch chip. *IEEE Journal on Selected Areas in Communications* 9 (8): 1265–1279.

Nicol, D., and G. Yan. 2006. High-performance simulation of low-resolution network flows. *Simulation* 82 (1): 21.

Stiliadis, D., and A. Varma. 1996. Design and analysis of frame-based fair queueing: A new traffic scheduling algorithm for packet-switched networks. *SIGMETRICS Performance Evaluation Review* 24 (1): 104–115.

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

Zhang, L. 1991. VirtualClock: A new traffic control algorithm for packet-switched networks. *ACM Transactions on Computer Systems (TOCS)* 9 (2): 124.

Zukerman, M., T. Neame, and R. Addie. 2003. Internet traffic modeling and future technology implications. In *IEEE INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*.

AUTHOR BIOGRAPHIES

DONG JIN is a Ph.D. student in the Department of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign. He holds a B.Eng. with first class honors in computer engineering from Nanyang Technological University, Singapore (2005), and a M.S. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign (2010). His research interests lie in the areas of computer security, large-scale computer and communication system modeling and simulation. His email address is <dongjin2@illinois.edu>.

David M. Nicol is Professor of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign. He holds a B.A. in mathematics from Carleton College (1979), and M.S. and Ph.D. degrees in computer science from the University of Virginia (1983,1985). Prior to joining UIUC, he taught at the College of William & Mary, and Dartmouth College. He has served in many roles in the simulation community (e.g. Editor-in-Chief of ACM TOMACS, General Chair of the Winter Simulation Conference Executive Board of the WSC), was elected Fellow of the IEEE and Fellow of the ACM for his work in discrete-event simulation, and was the inaugural recipient of the ACM SIGSIM Distinguished Contributions award. His current research interests include application of simulation methodologies to the study of security in computer and communication systems. His email address is <dmnicol@illinois.edu>.