Empirical TCP Profiles and Application *

Catalin T. Popescu *A. Udaya Shankar*
cpopescu@cs.umd.edu shankar@cs.umd.edu
Department of Computer Science
University of Maryland, College Park

May 3, 1999

Abstract

We characterize a TCP implementation by a function, called a profile, that expresses the instantaneous throughput at the source in terms of the instantaneous roundtrip time and instantaneous loss rate for bulk transfers. We *empirically* obtain profiles of several TCP implementations, accurately enough to distinguish not only the TCP version but also the implementation (BSD, Windows, etc).

Profiles have several uses: comparing different TCP implementations, diagnosing a TCP implementation, quantifying TCP-friendly flows, etc. We devise a method that uses profiles to compute the time-evolution of instantaneous performance metrics (throughput, queue size, loss rate, etc.) of TCP networks. Comparison against *ns* simulations shows the method to be accurate and fast.

1 Introduction

Consider a TCP connection doing a bulk transfer from a source host to a destination host that is always ready to receive. The current throughput at the source depends on its current window size and retransmission timeout, which in turn depend on the current roundtrip time (RTT) estimate, the current packet loss estimate, the congestion control algorithm (Tahoe, Reno, Vegas, etc.), and the particular implementation (BSD Unix, Linux, Solaris, Windows NT, Windows 95, etc.). The RTT estimate is obtained from (exponential averaging of) measured packet RTTs. The packet loss estimate is based on timeouts and duplicate ack reception; each packet loss estimate results in a retransmission.

Thus for a TCP source implementation, we expect the *instantaneous* throughput to be a function of the *instantaneous* RTT and *instantaneous* estimated packet loss rate, We refer to this function as the **profile** of the TCP source implementation. We refer to these metrics as instantaneous because they are computed over a small number of roundtrip times.

*This work is supported partially by ARPA contract number DABT6396C0075 and DoD contract number MDA90497C3015 to the University of Maryland. It should not be interpreted as representing the opinions or views of ARPA, DoD, or the U.S. Government.

The importance of such a characterization of TCP is well recognized. It serves to compare the performance of different TCP versions and implementations. It provides a reference for diagnosing and debugging TCP implementations. It provides a quantitative criteria for deciding the TCP-friendliness of a flow. In this paper, we show how profiles can be used to compute instantaneous performance metrics (throughput, queue size, loss rate, etc.) of TCP networks.

Most investigations of TCP performance has been done through analytical models [1, 2, 7, 8, 10]. The idea of obtaining the throughput of a connection as a function of encountered roundtrip time and loss rate is not new. References [8, 10] develop analytical models to obtain this function. Reference [10] models most of the details of TCP Reno except for fast recovery, and analytically develops the dependence between *long-term* throughput, roundtrip time and estimated loss rate.

However, in order to obtain tractable analytical models, many simplifications are required, making the accuracy and utility of the model questionable. Moreover, many implementations of TCP Reno differ from the TCP Reno specifications [5, 6, 12], for example, in how they manage timers, how they interpret the specifications, and so on [3, 11]. All these can induce large differences between an analytical model and reality.

Consequently we obtain TCP profiles *empirically*, by analyzing packet traces at the network-interface (e.g. Ethernet) level of TCP source implementations. This yields much more accurate profiles, accounting for all source implementation details. Indeed the accuracy is more than enough for distinguishing different TCP implementations and for doing fault diagnosis. For example, the profiles for NetBSD 1.2 and Windows NT 4.0 SP3 show that NetBSD is up to three times faster than Windows NT 4.0 SP3 for low loss rates, whereas Windows NT is 30 to 40% faster for loss rates between 0.1 and 0.25. As a matter of fact, both implement TCP Reno. The profile for SunOS 5.5 brings out the well-known problem of low RTO initialization, described in [3, 11] (Sun has fixed it in the next version).

Accuracy in profiles is essential if they are to be used for differentiation or diagnosis of TCP implementations. It is also essential if they are to be used to model TCP sources for evaluating performance metrics of a network. Inaccurate profiles result in inaccurate metrics, as we will see later.

In this paper, we present a method that uses profiles to compute the time-evolution of instantaneous performance metrics (throughput, queue size, loss rate, etc.) of TCP networks. We validate our method against simulations performed with the *ns* [4, 9] simulator. Our method proves to be both accurate and fast.

Our method is based on numerical evaluation of time-dependent queuing models. Given a TCP network to evaluate, we model the network by a queuing network in the usual way, with one queue for each outgoing link. Each TCP source is modeled as a time-dependent stochastic process whose current rate is the instantaneous throughput indicated by the appropriate TCP profile for the current network state. We solve the queuing network numerically, using certain approximations, to obtain the time evolution of instantaneous metrics, specifically, the average queue size, loss probability, and utilization at time t, for every connection at every link.

The rest of the paper is organized as follows. In Section 2 we describe the experimental procedure for obtaining TCP profiles and present profiles for NetBSD 1.2, Windows NT 4.0 (SP3), and SunOS 5.5. In Section 3 we describe the method for computing the instantaneous metrics of a TCP network. In Section 4 we apply the method to several networks and compare the results against *ns* simulations. The networks range from small to large (100 nodes, 120 links, link bandwidth of 10^4 packets/sec, link buffer of 10^4 packets). Section 5 concludes the paper.

2 TCP Profiles

Obtaining the profile of a TCP source implementation involves the following: choose a host which runs the target implementation to act as the source, choose a set of hosts in the Internet to act as destinations, and trace a number of bulk transfers between the source and these destinations. Processing the resulting traces yields a set of points in the space of *instantaneous throughput*, *instantaneous roundtrip time* and *instantaneous loss rate*. Finally we fit a surface through these points.

Throughout, we use *milliseconds (ms)* for time measurements and *packets/millisecond* for throughput.

Before giving details of obtaining profile, we note some important points of our experiments:

- In this paper, we are interested in wide-area connections, going through several hops, rather than local-area connections. Roundtrip times in our experiments are typically over 60 ms.
- It is important that the set of source-destination paths explored in our experiments exhibit a wide range of loss rates and roundtrip times. Otherwise the resulting set of points will have large holes, leading to inaccurate surface fitting.
- We find in the wide-area context that the profile of a TCP source does not depend on the receive window advertised by the destination (provided of course that it is non-zero). That is, the limits imposed on the congestion window, hence throughput, by the roundtrip time and loss rate completely dominate the limit imposed by the destination receive window. The destination receive window would limit the throughput only in the case of very low roundtrip times (1 - 20 ms).
- We observe that the profile of a TCP source is not affected by the locations of the source and destinations, or how the hosts are connected to the Internet (LAN, modem, etc.). These factors affect where in the profile a connection spends most of its

time, but they do not affect the profile itself, i.e., do not affect the relationship between instantaneous throughput, instantaneous roundtrip time, and instantaneous loss rate.

- The profile of a TCP source is also not affected by the the TCP version and implementation of the destination. For example, a destination that delays acks, or acks every other packet only, induces an increase in the instantaneous RTT estimate at the source, which as we shall see decreases the throughput. However, the profile of the source is itself not affected.
- Because we are interested in wide-area connections, where the roundtrip time is usually over 60 ms, we do not consider the operating system overhead or processor speed as limiting factors. The delays they introduce are at most 1 or 2 ms, which we think is negligible considering the round-trip times.

2.1 Obtaining the Traces

We chose the hosts from Table 1 as the sources for which to obtain TCP profiles (Windows NT 4.0 from Service Pack 2 onwards implements the complete TCP Reno). We chose the hosts from from Table 2 as the destinations to which the sources connect.

Host Name	IP Address	Operating System
skinner	128.8.129.33	Windows NT 4.0 Service Pack 3, Intel Platform
homer	128.8.129.183	NetBSD 1.2
yangtze	128.8.128.18	SunOS 5.5

Table 1: Source hosts (full host ids are omitted for refereeing purposes)

Destination Name	IP Address	skinner	homer	yangtze
deepthought.cs.utexas.edu	128.83.120.2	-	36	24
joplin.cs.ucsb.edu	128.111.49.134	30	36	24
nimba.ccs.neu.edu	129.10.116.126	30	36	24
lava.cs.unh.edu	132.177.4.30	30	36	24
cwis.isu.edu	134.50.254.3	-	36	24
cat.isi.edu	128.9.160.205	30	36	24
nic1-4.hawaii.net	128.171.94.114	-	36	24
dir.mcc.ac.uk	130.88.200.4	-	36	24
nenya.hut.fi	130.233.224.36	-	36	24
sparrow.aup.fr	193.105.52.5	-	36	24
disco.cs.pub.ro	141.85.37.9	30	36	24
happy.snu.ac.kr	147.46.114.110	30	36	24
aljahedh.uaeu.ac.ae	194.69.1.68	-	36	24

Table 2: The destinations and the number of connections for each pair source/destination

Each connection consists of a 1.5MB transfer from source to destination, specifically to port 9, the discard port. The measurements were done at morning, mid-day, afternoon, evening and night, and on weekdays and weekends. The packets were traced at the Ethernet level using *tcpdump*.

2.2 Processing the Traces

We process the traces by considering every sequence of k packets sent by the source. For each sequence, we obtain the estimated loss rate as the number of retransmitted packets divided by the total number of packets (k), the instantaneous RTT as the average RTT for all k packets, and the instantaneous throughput as the number of packets (k) divided by the time spent in sending them. This triplet gives us one sample point in the space of instantaneous throughput, instantaneous roundtrip time, and instantaneous loss rate.

We varied k between 50 to 1000 packets, and did not find significant differences in the resulting profiles for the different values of k. Thus we conclude that the instantaneous metrics of a TCP connection become stable over 1-sec intervals (roughly corresponding to 50 packets) and do not change over 200-sec intervals (roughly corresponding to 1000 packets).

For $k = 100$, we plot, in Figure 1, the sample points obtained for NetBSD and Windows NT at loss rate of 0.01 and 0.1 respectively. In Figure 2, we plot the same points, using a logarithmic scale for the axes.

We easily observe that the points form a curve. Despite the fact that the sample points have more spread for NetBSD at a loss rate of 0.01, we can still observe the curve described by the sample points.

To see that the throughput measured in packets per second is not affected by the advertised windows size or by packet size, we present in Figure 3 the results observed, at a loss rate of 0.1, for two different destinations, cat.isi.edu and nimba.ccs.neu.edu. For cat.isi.edu, the advertised window size is 32736 bytes and the maximum segment size is 1514 bytes, whereas for nimba.ccs.neu.edu, the window size is 4096 bytes and the maximum segment size is 590 bytes.

2.3 Obtaining the Profiles

Next, we quantify the horizontal plane of instantaneous loss rate ($lrate$) and instantaneous roundtrip time (rtt) into rectangles of 0.01 on the loss rate axis by 10ms on the roundtrip time axis. For each rectangle we compute the instantaneous throughput ($throughput$) by averaging the throughputs of the sample points whose $lrate$ and rtt lie inside the rectangle.

The resulting surfaces are presented in Figure 4. These "raw" profiles look much more discontinuous than they actually are because of many "holes" in the horizontal plane, i.e., rectangles with no sample points.

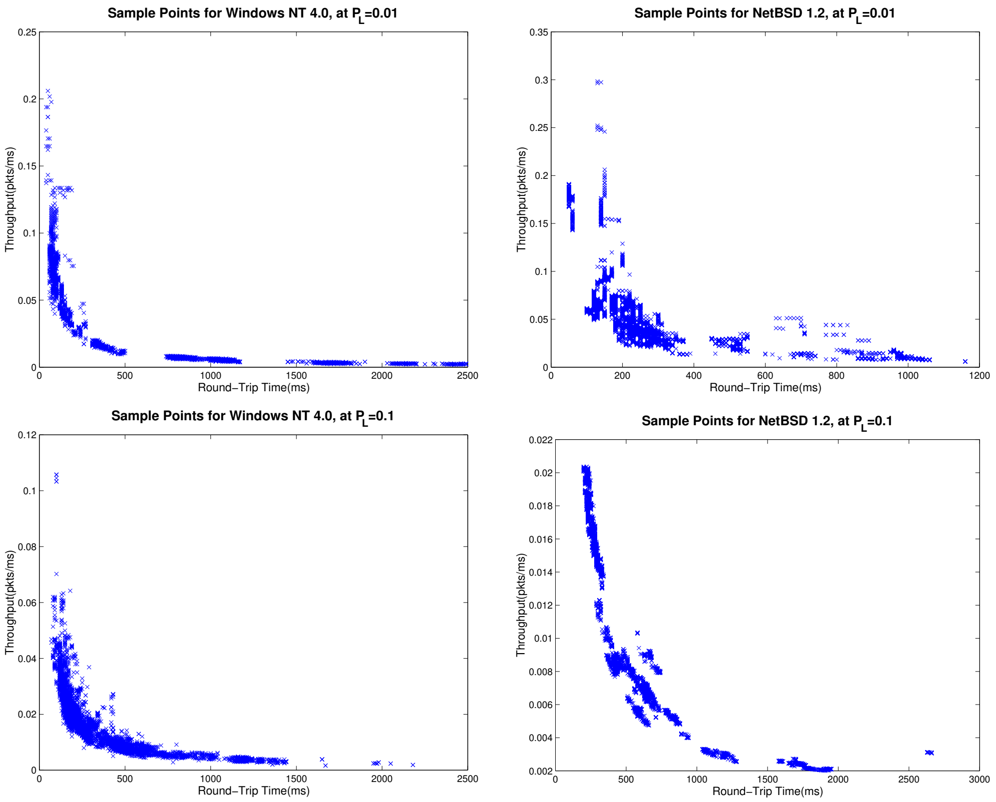


Figure 1: Sample points for Windows NT 4.0 SP3 (left) and NetBSD 1.2 (right), for loss probabilities (P_L) of 0.01 and 0.1 respectively.

The final profiles are obtained by smoothing these surfaces and filling in the holes. For each surface, we are looking for an approximating continuous function:

$$throughput = F(lrate, rtt) \tag{1}$$

We consider 'slices' through the surface for every quantified value of the loss rate. For each slice we find a corresponding approximating function.

In the logarithmic plots of the slices, the sample points describe a curve that is very close to a straight line. We found, using least square approximations, that polynomials of degree one are sufficient to closely approximate the curves in the logarithmic space for Windows NT 4.0 SP3 and SunOS 5.5 implementations. So, the approximating profiles for these two implementations has the form:

$$F(lrate, rtt) = B(lrate) * rtt^{-A(lrate)} \tag{2}$$

We used the first order least-square approximation in the logarithmic plane, and then we smoothed the resulting parameters for different values of $lrate$. For Windows NT 4.0 SP3 and SunOS 5.5, the values of parameters $A(lrate)$ and $B(lrate)$ are presented

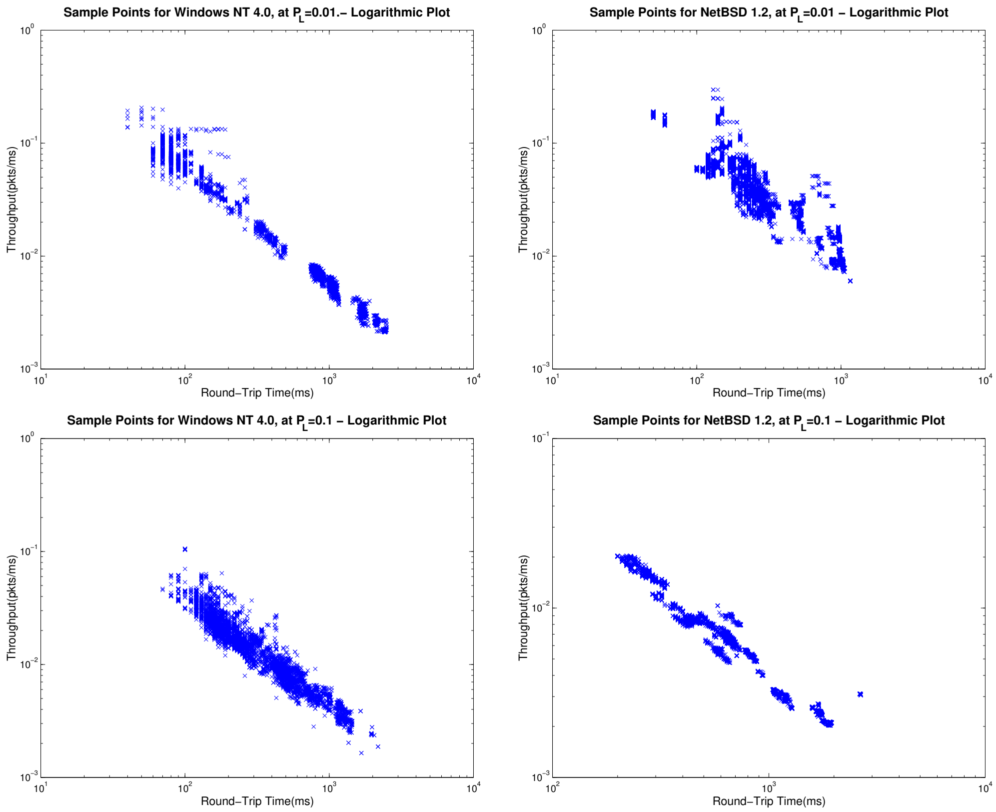


Figure 2: Sample points for Windows NT 4.0 SP3 (left) and NetBSD 1.2 (right), for loss probabilities (P_L) of 0.01 and 0.1 respectively. Logarithmic plots.

in Table 3, columns 2-5.

For NetBSD 1.2, a polynomial of degree one is not good enough for high loss rates. We have found the following approximating expression to be adequate:

$$F(lrate, rtt) = B(lrate) * rtt^{(A(lrate) - 10^{-4} * rtt * lrate)}. \tag{3}$$

where $A(lrate)$ and $B(lrate)$ are presented in Table 3, columns 6-7.

The resulting profiles for each implementation are plotted in Figure 5. By using the parameters from Table 3, and the appropriate expression, one can obtain accurate profiles for loss rates between 0.00 and 0.30 and roundtrip times between $60ms$ and $5000ms$.

2.4 Observations

From the profiles, we can easily remark the big differences between the throughputs achieved by different implementations. Comparing, for example, the profiles for NetBSD and Windows NT (see Figure 6(a)), we see that for low loss rates, NetBSD is up to three times faster than Windows NT. However, for higher loss rates, NT starts to be more

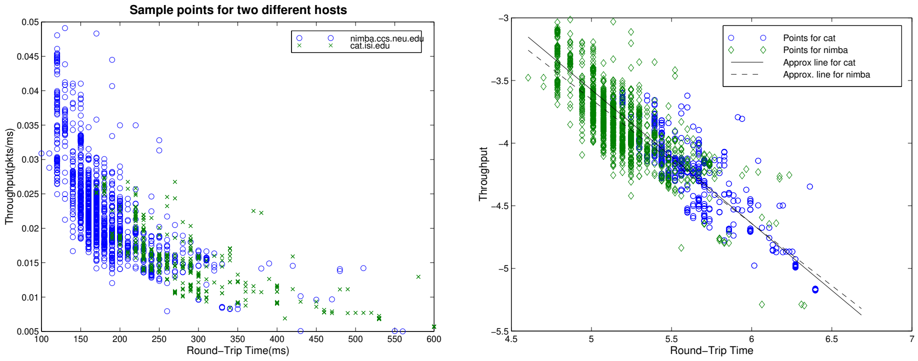


Figure 3: Comparison between sample points for cat.isi.edu and nimba.ccs.neu.edu, for a loss rate of 0.1. (Windows NT 4.0) - linear and logarithmic plots

efficient, and for estimated loss rates of 0.1 to 0.25 and roundtrip times over 300ms, Windows NT 4.0 SP3 performs 30 to 40% faster than NetBSD 1.2. We conclude that the differences are not induced by operating system overheads, because the RTTs are large compared to packet processing times. More likely, these differences are introduced by timer management and other implementation differences. The comparison between SunOS 5.5 and Net BSD 1.2 is given in Figure 6(b).

The differences between these profiles are so large that we wonder whether the overall performance in the Internet can be improved merely by just implementing TCP more carefully, for example, an implementation whose profile is the minimum of the three profiles for each value of $lrate$ and rtt. Recall that all three systems implement TCP Reno.

Another interesting fact can be observed if we plot the measurement points for SunOS 5.5. In Figure 7 we present the plots for a loss rate of 0.19. We observe the existence of some points placed far above the big mass of measurement points.

We analyzed the traces and concluded that this happens because SunOS 5.5 initializes the RTO at a very low value, and, even more strangely, sometimes retransmits packets that are already acknowledged. These problems were already pointed out in [3, 11]. This observation suggests to us that the profiling process is useful in debugging TCP implementations, in addition to quantifying their efficiency.

3 Fast Evaluation of Instantaneous Metrics of TCP Networks

By using profiles together with flow-based methods to solve time-dependent queuing networks, we have devised a method that computes the time-evolution of instantaneous performance metrics (throughput, queue size, loss rate, etc.) of TCP networks. We

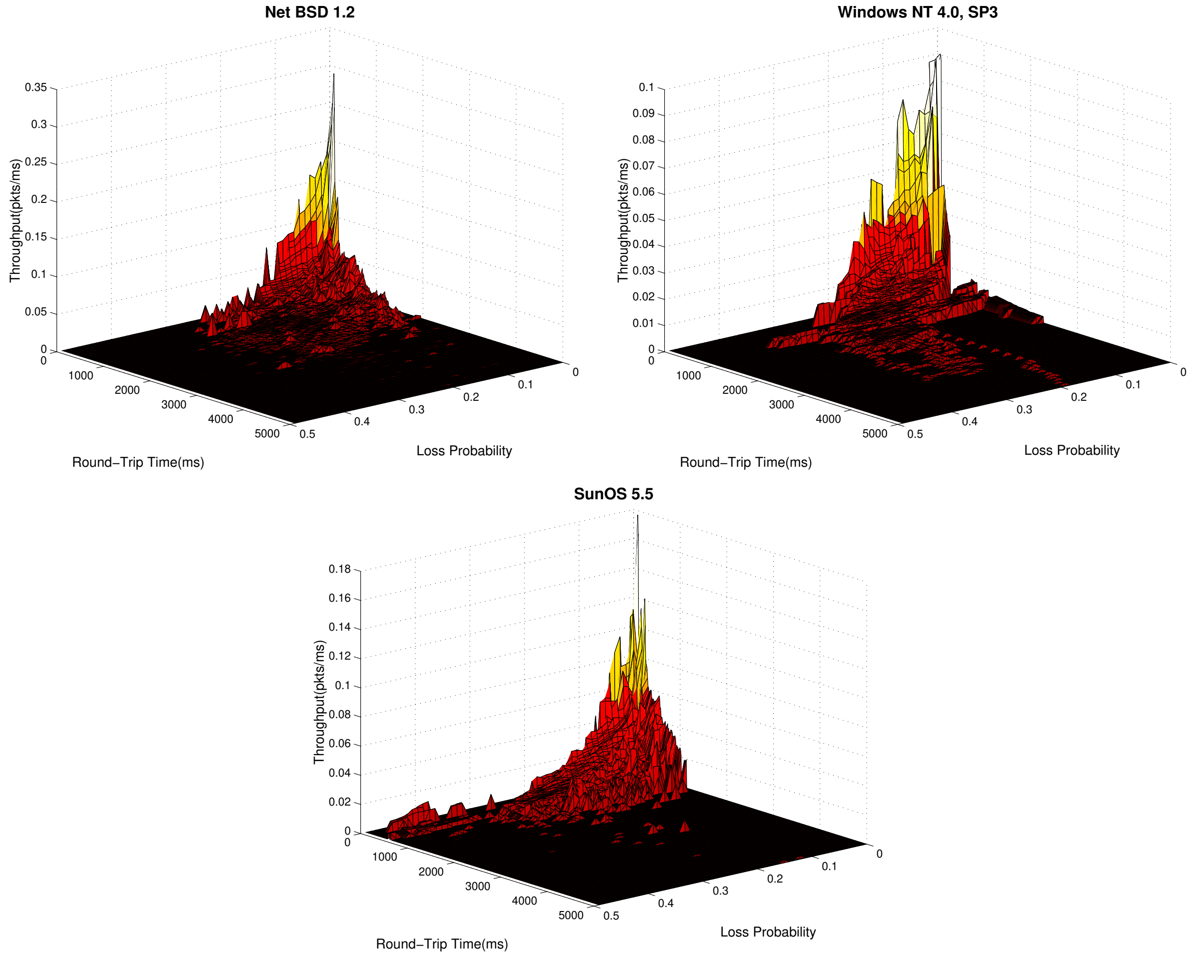


Figure 4: The rough resulting profile surfaces for NetBSD 1.2, Windows NT 4.0 and SunOS 4.4.

have compared our method against *ns* [4, 9] simulations, and find that it produces close results for most of our test cases.

3.1 Time-Dependent Queuing Network Model

We consider current-style TCP/IP networks, i.e., hosts and routers interconnected by links in some arbitrary topology. We assume that routers use Drop-Tail FIFO queues for outgoing links. We assume that the traffic of the TCP/IP networks is generated by bulk TCP connections. Each connection has a source, a destination, a start time, and a duration. The routes taken by the packets can be fixed or dynamically updated.

We model the TCP/IP network by a network of queues. A queue is introduced for every outgoing link of a router, and it models the system around the outgoing link. The packets are the customers of the queue. The link speed determines the service rate of the queue. The memory allocated by the router for the outgoing link determines the buffer space of the queue.

$lrate$	A_{NT}	B_{NT}	A_{SunOS}	B_{SunOS}	A_{NetBSD}	B_{NetBSD}
0.00	1.0305	6.780	0.7600	3.600	1.1000	25.000
0.01	1.0240	6.528	0.8100	4.000	1.0600	13.500
0.02	1.0173	6.165	0.8600	4.400	1.0270	8.100
0.03	1.0114	5.790	0.9050	4.850	1.0135	6.600
0.04	1.0050	5.430	0.9250	4.730	1.0050	5.700
0.05	0.9980	5.090	0.9340	4.550	0.9980	5.130
0.06	0.9910	4.750	0.9400	4.400	0.9910	4.750
0.07	0.9830	4.380	0.9470	4.300	0.9830	4.380
0.08	0.9755	4.045	0.9530	4.200	0.9760	4.050
0.09	0.9682	3.710	0.9590	4.120	0.9685	3.730
0.10	0.9607	3.430	0.9650	4.050	0.9607	3.430
0.11	0.9535	3.180	0.9700	4.000	0.9535	3.180
0.12	0.9445	2.955	0.9750	3.950	0.9445	2.955
0.13	0.9357	2.750	0.9800	3.900	0.9357	2.730
0.14	0.9267	2.530	0.9850	3.850	0.9267	2.520
0.15	0.9174	2.340	0.9900	3.800	0.9174	2.340
0.16	0.9100	2.160	0.9950	3.750	0.9100	2.160
0.17	0.9022	1.970	1.0000	3.700	0.9022	1.970
0.18	0.8941	1.800	1.0050	3.640	0.8941	1.800
0.19	0.8865	1.650	1.0100	3.570	0.8865	1.650
0.20	0.8800	1.530	1.0150	3.490	0.8800	1.530
0.21	0.8750	1.420	1.0200	3.420	0.8732	1.420
0.22	0.8710	1.340	1.0250	3.350	0.8667	1.340
0.23	0.8680	1.260	1.0300	3.300	0.8580	1.260
0.24	0.8660	1.190	1.0350	3.250	0.8490	1.190
0.25	0.8640	1.120	1.0400	3.200	0.8400	1.120
0.26	0.8620	1.090	1.0450	3.150	0.8300	1.090
0.27	0.8610	1.040	1.0500	3.100	0.8200	1.040
0.28	0.8605	1.020	1.0550	3.050	0.8100	1.020
0.29	0.8600	0.990	1.0600	3.000	0.8000	0.990
0.30	0.8597	0.970	1.0650	2.950	0.7900	0.970

Table 3: Profile Parameters

Each TCP source generates packets according to a time-dependent stochastic process whose rate is varied with time. The rate variation is determined by the current network state and the profile characterizing the TCP source implementation. Specifically, from the current network state and current path of the TCP connection, we obtain the loss rate and roundtrip time experienced by the connection. Plugging this into the profile of the TCP source yields its current rate.

In the TCP/IP network, the paths taken by the packets are specified by routing tables, and they may be fixed or dynamically updated depending on the routing scheme. This is captured in the model by having time-varying routing probabilities between queues whose current values are determined by the current distribution of packets in the queues, which in turn is determined by the current rates and current paths of the transport connections.

One way of determining the routing probabilities is as follows. For each queue i and

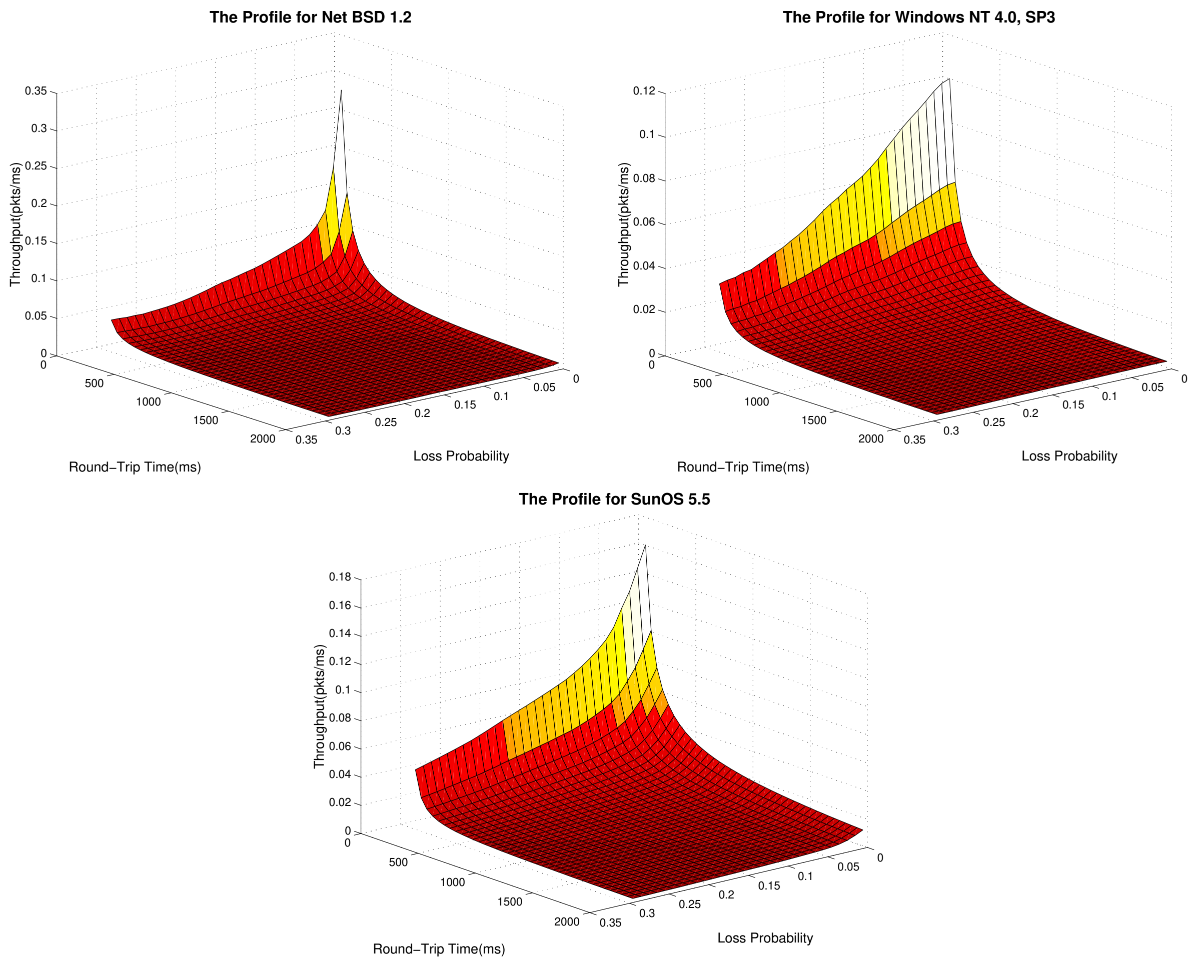
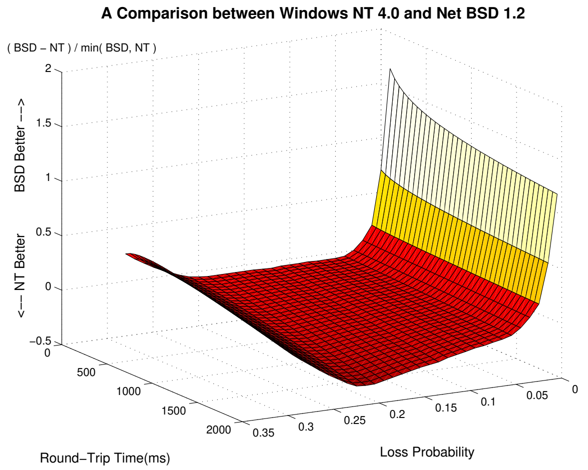


Figure 5: The TCP Profiles for NetBSD 1.2, Windows NT 4.0 and SunOS 5.5 .

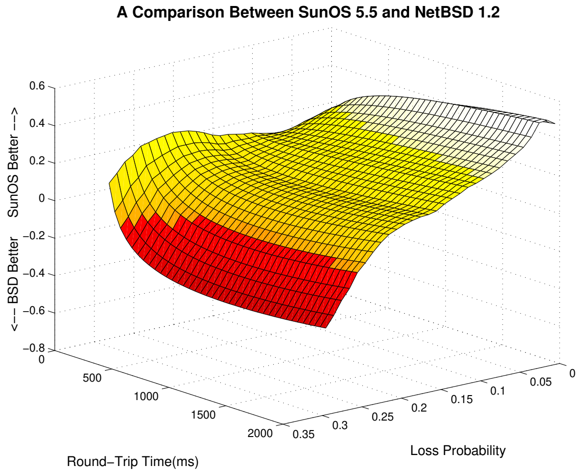
transport connection c, maintain the average number $N_{c,i}(t)$ of packets in the queue belonging to the transport connection at time t (our evaluation method computes these metrics). Then the probability that a packet leaving queue i at time t belongs to connection c is estimated by the fraction of connection c packets in queue i at time t, i.e., the ratio of $N_{c,i}(t)$ to $\sum_x N_{x,i}(t)$. This approach can be refined in several ways. For example, the packets in the queue can be grouped according to arrival time slots and maintain separate $N_{c,i}(t)$s for each group. Then the routing probabilities are based on the $N_{c,i}(t)$s for the group of packets at the head of the queue.

3.2 Comparing Our Method to Simulation Approaches

Our method models the TCP/IP network by a time-dependent queueing network and numerically computes the time-evolution of instantaneous ensemble metrics using certain approximations (described in following subsections). We next discuss the limitations and advantages of our method against that of using detailed packet-level simulators



(a)



(b)

Figure 6: A comparison between the TCP profiles of NetBSD 1.2, Windows NT 4.0 SP3 and SunOS 5.5

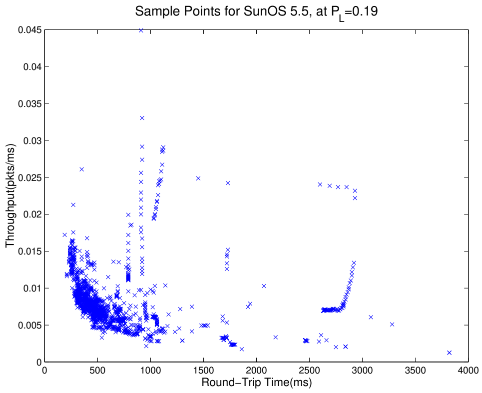


Figure 7: The sample points for SunOS 5.5, at a loss rate of 0.19

such as *ns*.

- In general, the instantaneous "sample path" metrics obtained from simulation runs will vary, sometimes considerably, around the instantaneous ensemble metrics obtained by our method. On the other hand, the computational cost of simulation is orders more than that of our method.
- Our method will capture TCP dynamics only partially. Specifically, it appears to accurately capture the effects induced by starting and ending TCP connections. However, it does not appear to accurately capture the perturbations caused by the synchronization between different TCP flows, the so-called phase effects, etc. We are currently working on ways to incorporate such effects in our method.

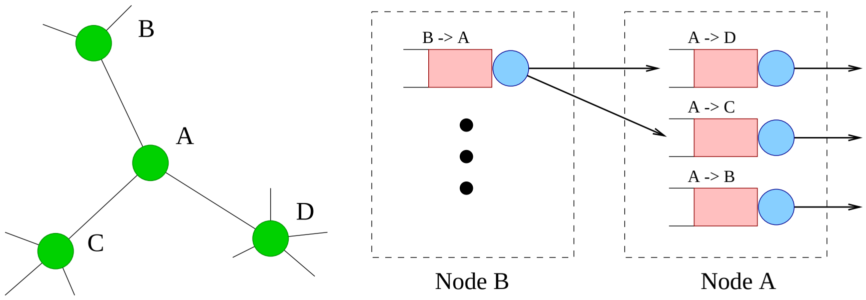


Figure 8: How the queuing model is constructed. Example.

- Our method tends to work better for larger networks (over 20 nodes) with many TCP connections, where the synchronization between TCP flows tends to be lower. Of course, packet-level simulation becomes significantly slower in these situations.
- Our method, as currently implemented, uses the simple ratio formula for estimating routing probabilities. This is not adequately accurate for very large networks (hundreds of nodes). We find that it becomes necessary to use the multiple-time-slot refinment in estimating the routing probabilites, which reduces the speed by approximately 30% This is work in progress. (Of course, packet-level simulation becomes prohibitively slow in these situations.)
- The size of the time increments in our numerical evaluation can be varied, allowing us to easily trade between compuation cost and accuracy. Such trade-offs are practically impossible to do in packet-level simulators.
- Just as different TCP implementations have different profiles, we found that TCP Reno in *ns* has its own profile, one that differs significantly from that of real TCP Reno systems. This suggests that perhaps stochastic models using empirically-obtained profiles give a more accurate measure of what is happening in reality, even if the results are different from the ones obtained by simulation.

3.3 Evaluation Method

The evaluation time is divided into small time steps. For each time step t, we determine, for each queue i, the instantaneous buffer occupancy (i.e., average queue size) $N_i(t)$, the instantaneous loss probability $B_i(t)$, and the instantaneous utilization $U_i(t)$. Based on these, the instantaneous roundtrip time and instantaneous loss rates of each TCP connection is computed. Plugging these into the TCP profile of the connection, we obtain the instantaneous rate of that connection for the next time step of evaluation.

An outline of the algorithm is presented in Figure 9.

Obtaining the time-evolution of $N_i(t)$, $B_i(t)$, and $U_i(t)$ is a difficult task. We describe this computation in the following paragraphs.

```
• compute routes in the network;
• initilaize network model state;
for t = startTime to endTime step deltaT do
    • update routes in the network
    foreach connection in TCP Connections Set do
            • compute rtt and lrate on the connection path
            • connection throughput = profile( rtt, lrate );
    end foreach;
    foreach queue i in network model do
            • compute N_i(t), B_i(t), U_i(t)
    foreach connection in TCP Connections Set do
            foreach queue i on connection path do
                    • compute N_{c,i}(t);
    • dump metrics for time t;
end for;
```

Figure 9: Overview of the evaluation method.

Let the evaluation time be divided into time steps of size δt. For a step starting at time t we compute, for each queue, the buffer occupancy at the end of that step, $N_i(t + \delta t)$, using the fluid flow formula:

$$N_i(t+\delta t) = N_i(t) + \begin{cases} [\lambda_i(t) - \mu_i(t)U_i(t)]\delta t & \text{if } \lambda_i(t) - \mu_i(t)U_i(t) < 0 \text{ or } N_i < K_i \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where $\lambda_i(t)$ is the input traffic rate for queue i at time t, $\mu_i(t)$ is the service rate for queue i at time t, and K_i is the buffer space for queue i.

For computing $U_i(t)$ we again use the formula from fluid flow approximation:

$$U_i(t) = \begin{cases} \lambda_i(t)/\mu_i(t) & \text{if } N_i(t) = 0 \text{ and } \lambda_i(t) < \mu_i(t) \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

For computing $B_i(t)$, we do not use the fluid flow formula because we found from experience that it does not yield correct results. Instead, we approximate $B_i(t)$ by the steady-state blocking probability of an $M/M/1/K_i$ queue with average queue size of $N_i(t)$.

The input to queue i consists of external sources and outputs of other queues. To compute the input rate $\lambda_i(t)$ of queue i, we employ the decomposition approximation. A departure from queue i is routed to queue j with a time-dependent probability $r_{ij}(t)$, and leaves the network with probability $1 - \sum_j r_{ij}(t)$. The arrivals to queue i consist of external arrivals $\lambda_i^*(t)$ (from outside the network) and departures from queues in the network routed to queue i. The total arrival rate of queue i, $\lambda_i(t)$, is given by:

$$\lambda_i(t) = \lambda_i^*(t) + \sum_{j=1}^{n} r_{ji}(t)\ \mu_j(t)\ U_j(t). \tag{6}$$

To compute $N_{i,c}(t)$, we use a difference formula:

$$N_{i,c}(t + \delta t) = N_{i,c}(t) + [\lambda_{i,c}(t) - \lambda_{j,c}(t)]\delta t \qquad (7)$$

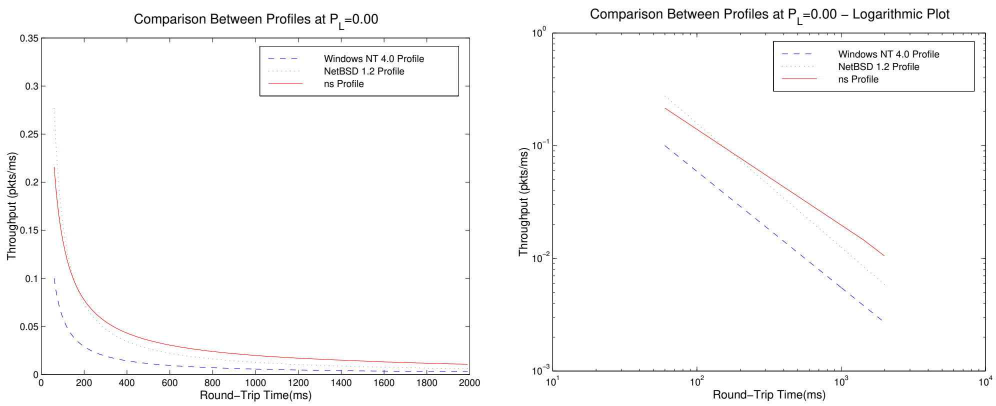
where $\lambda_{i,c}(t)$ is the rate of class c at queue i, and j is either the next queue on the path of connection c or a fictional sink queue if i is the last queue on the path.

4 Example Evaluations

We applied the evaluation method outlined above to several TCP networks, and compared the results against simulations obtained from the *ns*.

In the course of our comparison, we discovered that the profile for the TCP Reno implementation in *ns* differs from all of the profiles we have measured (i.e., Windows NT SP3, SunOS 5.5, NetBSD 1.2). Of course, this is not unexpected since *ns* most likely does not simulate many details of real implementations. This emphasizes even more the importance of the implementation details in TCP performance evaluation.

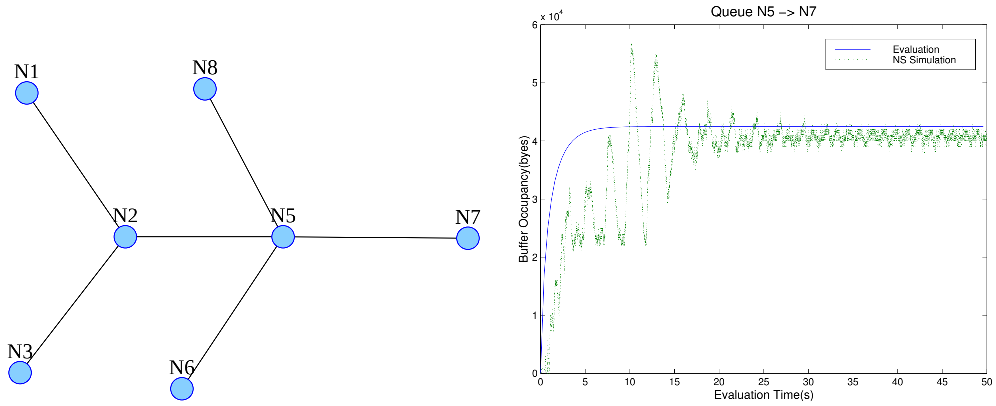
We obtained the profile for the *ns* implementation of TCP Reno, and used this profile in our evaluation method. A comparison between the profiles of Windows NT 4.0 SP3, NetBSD 1.2 and NS, for a loss probability of 0.00, is given in Figure 10.



(a) (b)

Figure 10: Comparison between NT, NetBSD and NS, for a loss rate of 0.00, in linear(a) and logarithmic(b) plane.

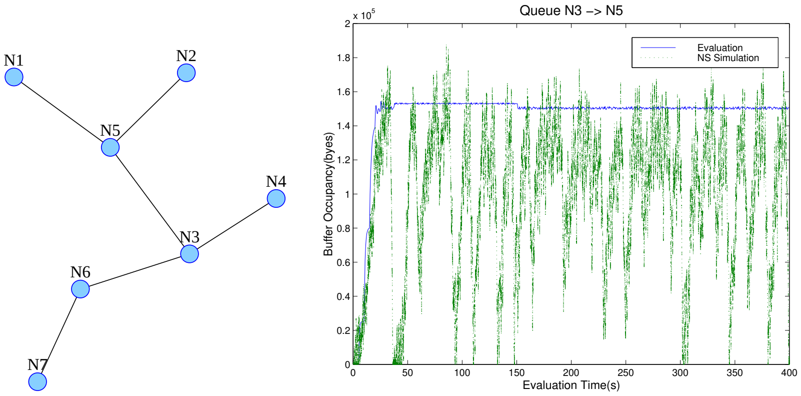
Example 1 The first example is the network from Figure 11(a). Here, we have four TCP connections from nodes N1, N3, N8 and N6 to N7. The queue size at the bottleneck link, N5 $\rightarrow$ N7 is presented in Figure 11(b) and is compared agains one *ns* simulation run. The run times for both methods are insignificant.



(a) (b)

Figure 11: Example 1: Small network, 4 TCP connections.

Example 2 In this example we have chosen the topology from Figure 12(a), but we have 66 TCP connections, with start times ranging between 0 sec and 20 sec and no stop time during the evaluation period. For this network, we plot the queue size of the most congested link, N3 $\rightarrow$ N5, in Figure 12(b). Comparing against the *ns* simulator, we observe that the phase effects which appear during the simulation is not captured by our evaluation method. However our method does compute the correct buffer occupancy during the time intervals when the phase effects do not affect the performance.



(a) (b)

Figure 12: Example 2: Small network, high load. Running Time: our method takes 2 sec; ns takes 22 sec when no metrics are dumped and 97 sec when all metrics are dumped.

Example 3 This is the medium size network from Figure 13(a). The load is moderate and the connections start, one by one, between time 0 sec and 30 sec. We plot the buffer occupancy for some queues of the network in Figure 13(b-d).

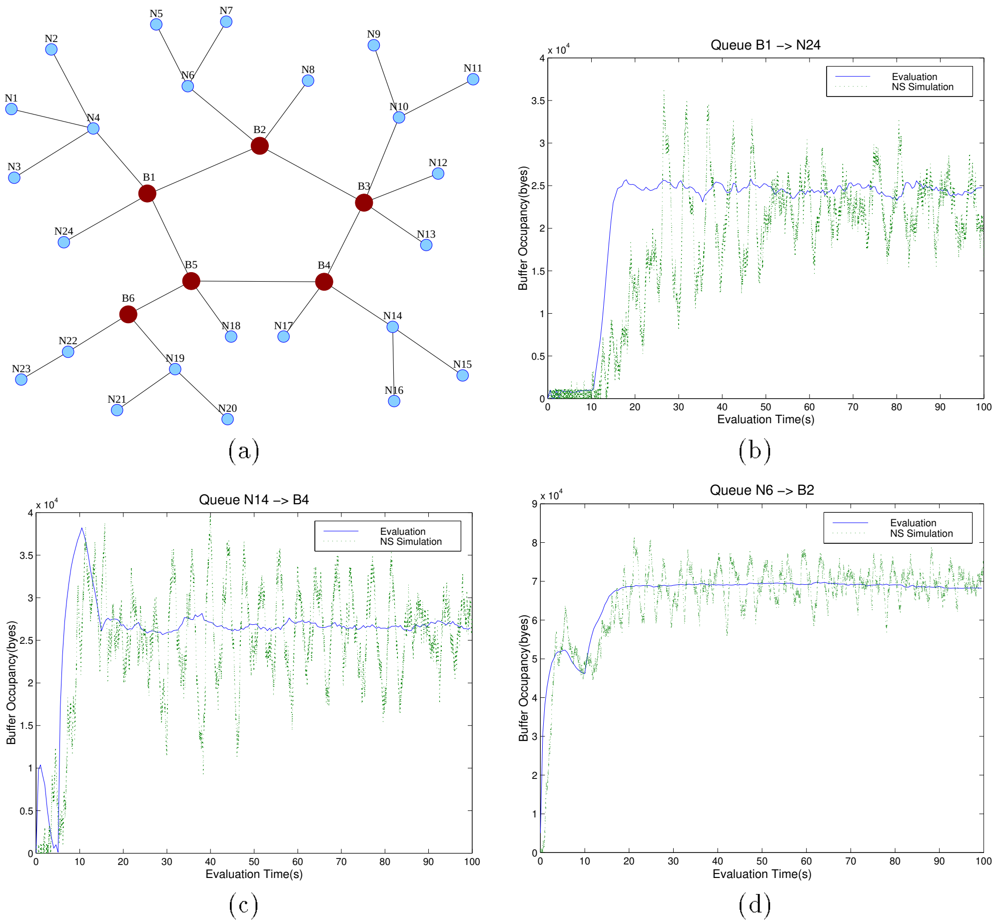


Figure 13: Example 3: Medium network, medium load. Running Time: our method takes 2 sec; ns takes 10 sec (no metrics dumped) to 60 sec (all metrics dumped).

Example 4 For this example we have a larger network, of about 100 nodes and medium load. Here, the connections come up and go down during the evaluation period. The topology is presented in Figure 14, and some plots are displayed in Figure 15.

Example 5 Here we use the same network as in the previous example, (Figure 14), but with higher load. The plots are presented in Figure 16.

5 Conclusions

We have shown that a TCP source implementation can be accurately characterized by an empirically-obtained profile, that is, a function expressing instantaneous throughput at the source in terms of instantaneous roundtrip time and instantaneous loss probability. We obtained TCP profiles for three commonly used operating systems, and conclude that the profiles are significantly influenced by implementation details.

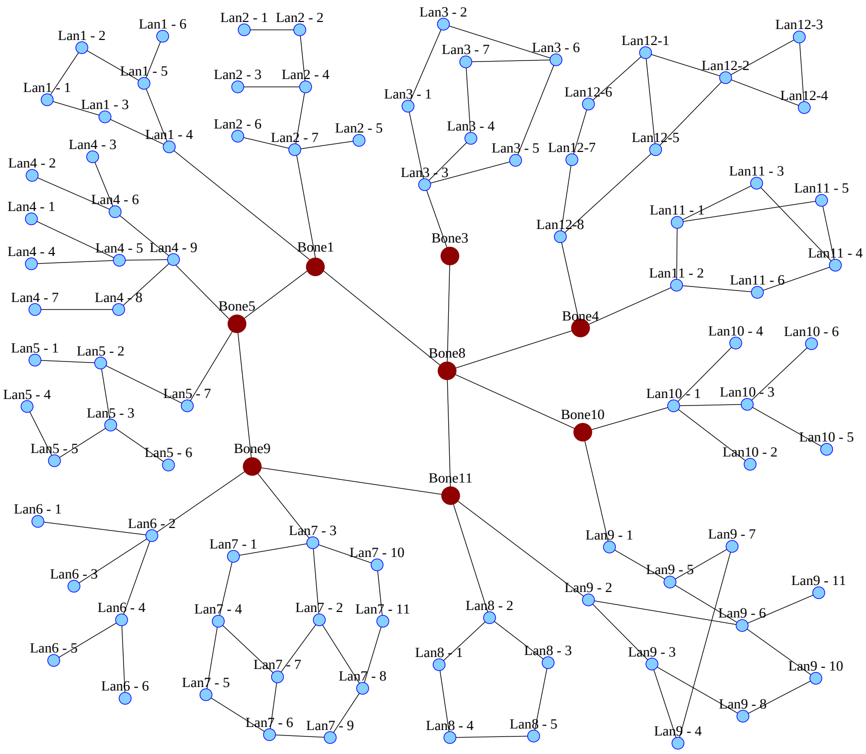


Figure 14: Example 4: Large Network - Topology.

Profiles have many uses, including debugging implementation problems, evaluating the efficiency of TCP implementations, quantifying TCP-friendly flows, and so on. We used these profiles to devise a method to evaluate the metrics of a network with many TCP connections. The method proved to be accurate and much faster than simulation. Its main drawback is that it misses the phase effects which sometimes appear in highly loaded networks. This happens because our method distributes equally the loss probability, at a queue, among the flows passing through that queue.

We stress the fact that the profile for TCP Reno in *ns* differs from all the profiles of real systems measured by us. We suggest that the empirically-obtained profiles give the real measure of what is happening in reality, even if the results are different from the ones obtained by simulation.

The size of the evaluation step is an important factor in achieving a good speed. We have obtained these results using steps of 0.05 seconds. Using a larger step size will result in slightly fluctuating metrics. However, the computation is faster using the bigger step, and, for very large networks, one can trade some precision for speed. Currently, we investigate the possibility of using variable step sizes. Such trade-offs are not at all easy to do with simulations.

Future work will involve an investigation for other uses of the TCP profiles. Possibilities include a definition of TCP-friendly UDP connections, prediction of the available

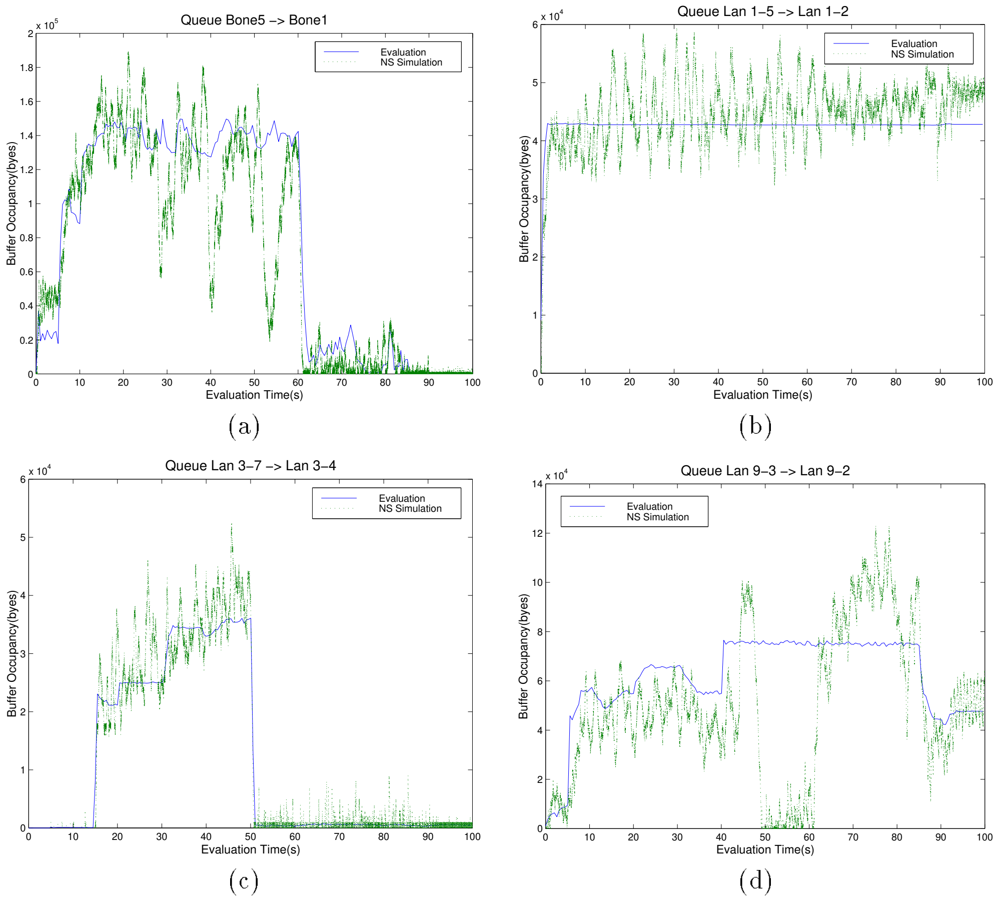


Figure 15: Example 4: Buffer occupancy for some queues. Running Time: our method takes 16 sec; ns takes 71 sec (no metrics dumped) and 937 sec (all metrics dumped).

throughput for adaptive network applications, determining the conformance of the flows passing through gateways, and so on.

We plan to extend the evaluation method to RED gateways and, also, to integrated-services TCP/IP networks. We also intend to increase the speed of our method by a more careful implementation of our evaluation tool, and by using variable step sizes.

References

[1] E. Altman, J. Bolot, P. Nain, D. Elouadghiri, M. Erramdani, P. Brown, and D. Collange. Performance Modelling of TCP/IP in a Wide-Area Network. Technical Report No. 3142, INRIA, 1997.

[2] P. Brown, D. Collange, and C. Fenzy. A Discrete Time Analysis of TCP in Presence of Cross Traffic. In *IEEE INFOCOM '95*, 1995.

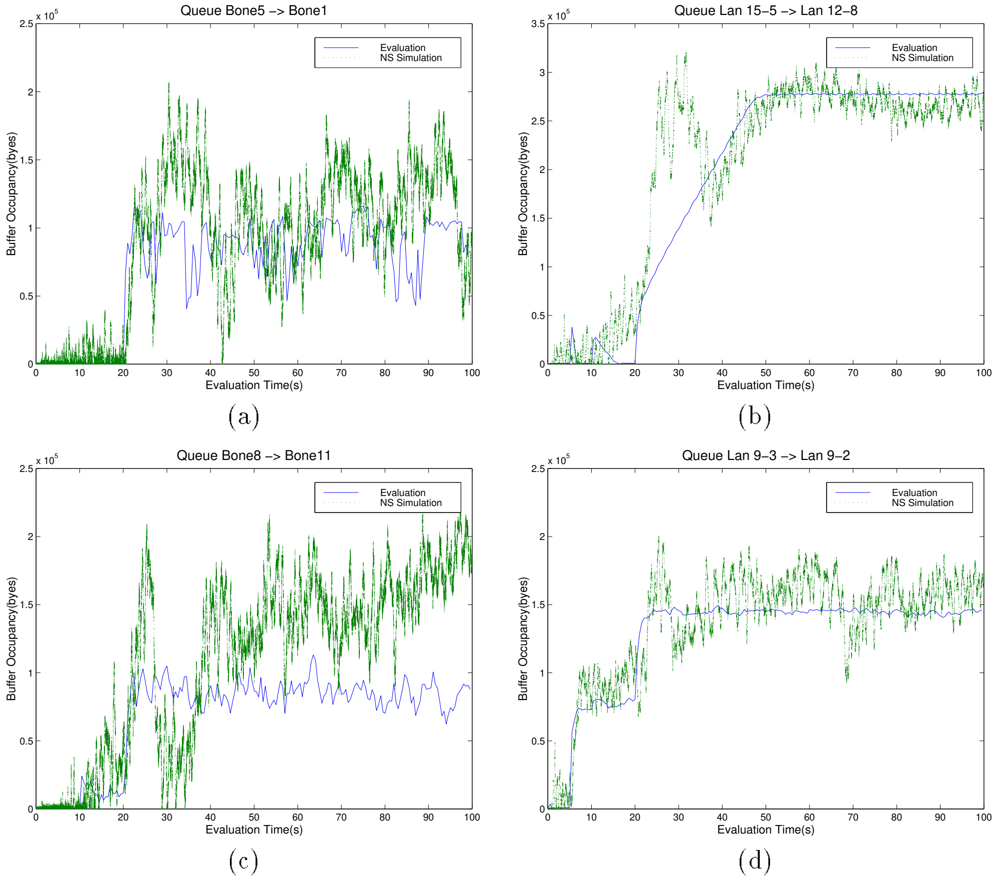


Figure 16: Example 5: Buffer occupancy for some queues. Running Time: our method takes 29 sec; ns takes 146 sec (no metrics dumped) and 2150 sec (all metrics dumped).

[3] D.E. Comer and J.C. Lin. Probing TCP Implementations. In *USENIX Summer '94 Conference*, 1994.

[4] K. Fall and K. Varadhan. *ns Notes and Documentation.*

[5] Information Sciences Institute - University of Southern California. *RFC 793: Transmission Control Protocol - Protocol Specification*, September 1981.

[6] V. Jacobsen, R. Braden, and D.Borman. *RFC 1323: TCP Extensions For High Performance*, May 1992.

[7] T.V. Lakshaman and U. Madhow. The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss. *IEEE/ACM Transactions on Networking*, 5(3):336–350, 1997.

[8] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. In *Computer Communication Review*, volume 27(3), July 1997.

[9] S. McCanne and S. Floyd. *ns - network simulator (version 2) - Manual Page.*

[10] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *ACM SIGCOMM '98*, 1998.

[11] V. Paxson. Automated Packet Trace Analysis of TCP Implementations. In *ACM SIGCOMM '97*, 1997.

[12] W. Stevens. *RFC 2001: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*, January 1997.