On Internet Path Modeling

Kaushik Lakshminarayanan
kaushik@cs.cmu.edu

Dongsu Han
dongsuh+@cs.cmu.edu

Samir Sapra
ssapra@cs.cmu.edu

Srinivasan Seshan
srini@cs.cmu.edu

ABSTRACT

In order to develop a robust distributed application, developers need realistic (overlay) testbeds, simulation tools or emulation testbeds. While overlay testbeds such as PlanetLab and RON give realistic network conditions, they don't lend themselves to repeatability and provide less control over experiments. On the other hand, the current trend in simulation (which gives repeatability) is to more accurately reproduce network topologies and scale to a large number of nodes. Emulation testbeds such as Emulab and ModelNet provide repeatability and more control, but not realistic network conditions.

We argue that for emulating the Internet, we need to be able to model the end-to-end characteristics of Internet paths. To this end, we propose a scheme to model cross-traffic for any given Internet path. Our approach uses a model of TCP with n flows to characterize the cross-traffic on the path using an abstract topological model. We give a systematic method for predicting the number of flows in a simple case where all the flows have the same RTT. We go further to predict the parameter n/RTT_{eff} when the flows have different RTTs. Finally, we show that it is important to predict this parameter as it effectively captures the cross-traffic behavior (aggressiveness).

1. INTRODUCTION

The Internet has evolved to such an extent that it has become difficult to mimic its behavior. Much progress has been made to infer the Internet topology at both the AS level and the router level for the purpose of simulation. But unfortunately, distributed application developers do not have testbeds which satisfy the key properties of realism and repeatability necessary for developing and debugging. Network testbeds used in the networking and distributed systems communities typically belong to one of two categories: (a) *overlay testbeds* such as PlanetLab [7] and RON [1], which allow the experimenter to run the experiment over the Internet with the help of overlay nodes, and (b) *emulation testbeds* such as Emulab [11] and ModelNet [12] which allow the experimenter to run the experiment over a collection of machines by creating artificial network conditions as desired by the experimenter.

Recently, there has been some effort towards combining the benefits of the above mentioned testbeds using a testbed called Flexlab [9]. Flexlab has three different network models to characterize end-to-end Internet path namely, simple-static, simple-dynamic and Application Centric Internet Modeling (ACIM). Simple-static and simple-dynamic models just take account of available bandwidth and latency using the measurement data done statically and dynamically respectively, on PlanetLab hosts. ACIM tries to model the Internet as perceived by the application – the application's traffic on Emulab is reproduced on PlanetLab and the network conditions experienced by this traffic are in turn applied in Emulab. This approach does well in achieving the goal of accuracy. However, this method suffers from two disadvantages: (1) Since the application's traffic characteristics are used to get a path model for the emulator, a transport protocol change in the application would require developing the path model again, which defeats the objective of repeatability. (2) The method as such is a heavy-weight mechanism for getting the response of an Internet path.

We believe that the inference of path characteristics should be decoupled from the application behavior. But, this leads to a trade-off between accuracy and flexibility. While developing a model of a path is hard and might be inaccurate, it gives more flexibility and repeatability than what is achievable using an application centric view of the path. It has been observed that creating a general-purpose model for the Internet or simulating the Internet is impractical [3, 4]. However, often what is required in evaluation testbeds is the behavior of Internet paths. So, instead of getting a general-purpose model of the entire Internet, we attempt to develop a course-grained model of Internet paths alone. Using a combination of measurement and modeling, we propose a way to model TCP cross-traffic. Starting from a model which predicts the number of TCP cross flows given these flows have the same RTT (using results from [6]), we generalize to flows with arbitrary RTTs (but we predict a slightly different parame-

ter to be described). We further give simulation results which show that a mixture of n TCP cross-traffic flows with the same value of n/RTT_{eff} (where RTT_{eff} is the harmonic mean of the RTTs of the n TCP flows) gives similar path characteristics of loss rate and throughput.

The paper is organized as follows. We give our model for TCP flows of same RTT and varying RTTs in Section 2 and describe our experimental setup. Then, we present our results on prediction of the number of flows (for same RTT) and n/RTT_{eff} (for varying RTT) in Section 3. We also study the behavior of different numbers of flows with the same value of n/RTT_{eff} in Section 3. We then look at related work in Section 4 and give our conclusions and future directions in Section 5.

2. METHODOLOGY

Our goal here is to characterize an Internet path as succinctly as possible without reducing the fidelity by much. To do this we take the following approach. Based on existing behavioral models of TCP, we extract relevant parameters of the traffic we should find out to model the traffic on an Internet path. Then we derive an active measurement technique to infer those parameters for a given path by injecting end-to-end flows that compete with cross-traffic that we want to model in the bottleneck link. Finally we evaluate the technique using `ns-2` simulation and emulation on Emulab.

2.1 Assumptions

Modeling cross-traffic on an Internet path is not a trivial task. In this work, we divide the problem into pieces and tackle parts of it. Here are the assumptions we make to limit the scope of this work.

- For a given path, there is only one (primary) bottleneck link.
- The traffic on the bottleneck link comprises only of steady-state TCP flows.
- Traffic is stable during the measurement.
- The loss rate of the bottleneck link is less than 1% (no timeouts).
- Bottleneck link router uses Random Early Detection (RED) queue management.

For the rest of this work, we focus on modeling TCP traffic on the bottleneck link.

2.2 TCP model

We know that TCP throughput can defined as a function of round trip time and loss rate experienced by the flow [6].

$$B(p) = \frac{1}{E[RTT]}\sqrt{\frac{3}{2p}} + o(1/\sqrt{p}) \quad (1)$$

which approximated by

$$B(p) \approx \frac{1}{E[RTT]}\sqrt{\frac{3}{2p}} \quad (2)$$

where $B(p)$ is the bandwidth consumed by a flow with expected round trip time $E[RTT]$ and packet loss event rate (probability) p. First, we look at the case where all flows have the same round trip time.

2.2.1 Same RTT

In this case, since all flows experience the same loss rate and have the same RTT, throughputs of the flows are equal. Thus the following relations hold, where C is the bottleneck link capacity and n is the number of flows.

$$C = nB(p) \quad (3)$$

Since we can directly measure the loss rate and RTT from active measurement, we only need to infer n, the number of flows. To infer n, we add a single flow from the source to destination, measure the packet loss rate p_1, and then add a second flow and measure the loss rate p_2. Assuming unchanged aggregate bandwidth, we see that

$$(n+2)B(p_2) = (n+1)B(p_1) = C \quad (4)$$

Using Equation 2, we get

$$n = \frac{1}{\sqrt{\frac{p_2}{p_1}} - 1} \quad (5)$$

2.2.2 Different RTTs

Here, the cross-traffic flows have different (expected) RTTs, say τ_1, τ_2,..., τ_n. We then add flows as done above to get loss rates p and p'. Here we also measure the RTTs of the flows; let t_{ij} be the RTT of the i^{th} flow when j of our flows are present. Since we use RED queue management, the packet loss rates for all the flows are same. Note that the packet loss rates are same in the same RTT case even if we use drop-tail queues. When the RTTs of the cross flows are different there is not much sense in predicting the number of flows as we did in the same RTT case because we do not know the RTTs of individual flows. Hence, we predict the parameter n/RTT_{eff} (the effective RTT is the harmonic mean of the RTTs), which in some sense models the aggressiveness of the cross-traffic. Intuitively, we can see that the more the number of flows, the more aggressive is the cross-traffic. Also, the lesser the RTT of a flow, the more is the aggressiveness of the flow. We later present results to show how this parameter is important in determining the TCP throughput and loss rate experienced on a path.

Along the lines of Equation 3, for different RTTs we have,

$$C = \left(\left(\frac{1}{\tau_1} + \frac{1}{\tau_2} + \ldots + \frac{1}{\tau_n} \right) + \frac{1}{t_{11}} \right) \sqrt{\frac{3}{2p_1}} \quad (6)$$

which we can write as

$$C = \left(\frac{n}{RTT_{eff}} + \frac{1}{t_{11}} \right) \sqrt{\frac{3}{2p_1}} \quad (7)$$

where RTT_{eff} is the harmonic mean of the RTTs of the cross-traffic flows. Similarly, when we add another flow, we get

$$C = \left(\frac{n}{RTT_{eff}} + \frac{1}{t_{12}} + \frac{1}{t_{22}} \right) \sqrt{\frac{3}{2p_2}} \quad (8)$$

Thus, from Equations 7 and 8, we have

$$\frac{n}{RTT_{eff}} = \frac{\sqrt{\frac{p_1}{p_2}} \left(\frac{1}{t_{21}} + \frac{1}{t_{22}} \right) - \frac{1}{t_{11}}}{1 - \sqrt{\frac{p_1}{p_2}}} \quad (9)$$

2.3 ns-2 setup

We set up a dumbbell topology as in Figure 1. The speed of the bottleneck link is 50 Mbps and every other link has the speed of 100 Mbps. We set the latency of the bottleneck link to 20 ms and provision enough queue space to fully utilize the bottleneck link. We use `TCP/Sack1` implementation with RED queue management.

To generate the cross-traffic on the bottleneck link, we start a TCP flow from one intermediary node to another node across the bottleneck link. The start time is randomized and we wait for 10 seconds for stabilization. The number of cross-traffic flows is varied across different simulations. We use slightly different settings and methods for the same RTT and varying RTT cases.

2.3.1 *Same RTT*

In the same RTT case, the RTT of every flow is the same. Thus, the only parameter we have to infer is the number of cross-traffic flows. The latency for every link except for the bottleneck link is set to 1 ms. 10 seconds after the start of the cross-traffic, two end-to-end TCP flows are introduced with a 100 second time gap between their start times. We measure the loss rate p_1, the loss rate that the first end-to-end TCP flow experiences while it is in steady state before the introduction of the second flow, and p_2, the loss rate that the first and second flows experience after the second flow stabilizes. From p_1 and p_2, we infer the number of cross-traffic flows as given by Equation 2.

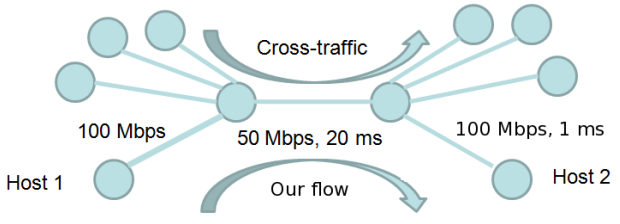

Figure 1: Simplified topology

2.3.2 *Different RTTs*

Here we simulate the case where the cross-traffic flows have different RTTs. The link from the cross-traffic end hosts to bottleneck routers are chosen from a uniform distribution of 1 ms to 10 ms. 10 seconds after the start of the cross-traffic, three end-to-end TCP flows are introduced with a 100 second time gap between their start times. We measure the loss rate p_1, the loss rate that the first end-to-end TCP flow experiences while it is in steady state before the introduction of the second flow, and p_2, the loss rate that the first flow and the second flow experience after the second flow stabilizes, and p_3, the loss rate that the three flows experience after the third flow stabilizes. From p_1, p_2 and p_3, we estimate n/RTT_{eff} (where RTT_{eff} is the effective RTT) with the help of Equation 9.

2.4 Emulab setup

Our goal with Emulab is to replicate our results from the simulator, while being one step closer to reality. As such, our Emulab setup mimics our `ns-2` setup, within the practical constraints of Emulab. Some differences are:

- The number of nodes available in Emulab is limited. Thus, our dumbbell topology includes exactly 2 competing node pairs. To emulate more than 2 TCP cross flows, we need to start multiple connections per node.
- The shortest latency supported in Emulab is 5ms. That is what we use for our short-latency links.
- Emulab does not support the `queue-limit` method of sizing queues. The default queue size provided is 50 slots, which is what we use in our experiment. Queue sizes may be specified in slots or bytes; queues may be sized up to 100 slots or 1 megabyte.

Further, to generate TCP flows, we use `iperf` in Emulab (this would correspond to `Agent/FTP` in `ns-2`). For tracing, we run `tcpdump` on the source node of our *introduced* TCP flow. This corresponds to monitoring transmission and reception events at the node *Host 1* in Figure 1. To parse the `tcpdump` output, compute TCP metrics, etc., `wireshark` and `tcptrace` are used.

3. EVALUATION

In this section, we present our results for both the same RTT case and the varying RTT case. First, we compare our predictions of the number of flows in the former case and n/RTT_{eff} in the latter case with actual values. Then, as an argument of how useful the prediction of n/RTT_{eff} is, we give results of TCP throughput and loss rate experienced on the path by varying n and RTT_{eff}.

3.1 ns-2 simulation

3.1.1 Same RTT

In this subsection, we briefly look at the results for the same RTT case before moving on the next section where we generalize our predictions to accommodate varying RTTs. Figure 2 shows how the packet loss rate varies with the number of cross-traffic flows. By curve-fitting, we find that the packet loss rate varies quadratically with the number of cross-traffic flows for small numbers of flows (here ~ 50) as has been observed in [5, 8]. Note that the loss rate exceeds 2% beyond 50 flows. This quadratic behavior validates Equation 5 that we use for predicting the number of TCP flows in the same RTT case.

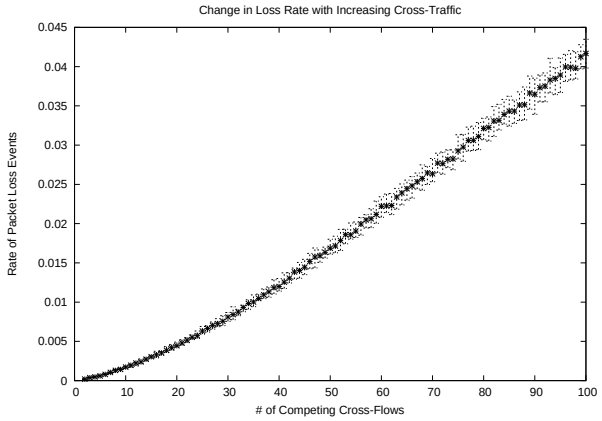

Figure 2: Quadratic behavior of TCP loss rate

Figure 3 shows the prediction of the number of flows. We see that the accuracy of our prediction is high for a small number of flows (< 13), above which the prediction is not as accurate and the error range increases. Nevertheless, our prediction still gives the number of flows within a factor of 1.5 for most flows. We do not include the results for n (number of flows) greater than 25, as the prediction sometimes gives an error of more than a factor of 2.

3.1.2 Different RTTs

For different RTTs, we use Equation 9 to predict n/RTT_{eff}. As described in the previous section, we add

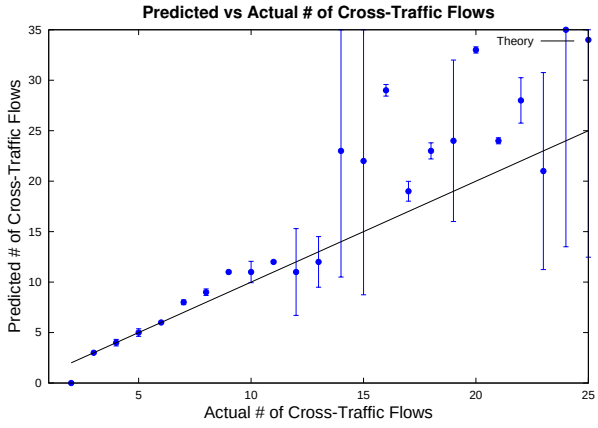

Figure 3: Same RTT case

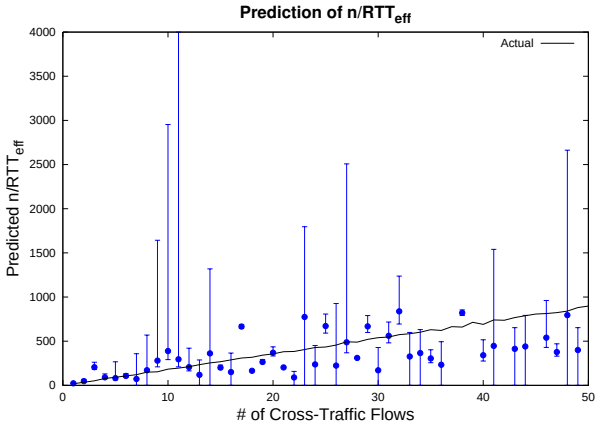

Figure 4: Different RTTs case

3 flows at regular time intervals to predict this value. Since marginal differences in bandwidth consumed and observed packet losses affect our prediction, we use the median of the 3 predictions obtained using Equation 9 to predict n/RTT_{eff}. Figure 4 shows our prediction of n/RTT_{eff} for different numbers of flows. We have removed from the graph those points that gave negative values of prediction. However, we observe that we can get better predictions if we use more of our flows to predict. We see that our prediction closely follows the actual values of n/RTT_{eff} for up to 30 flows and even beyond that, we are able to get fairly accurate estimates.

Although we have seen that we have been able to predict this magic parameter of n/RTT_{eff}, we have not yet seen how this parameter is useful in modeling an Internet path. In the next subsection, we shall see how this parameter models the aggressiveness of the cross-traffic and thus gives rise to similar path characteristics of TCP throughput and loss rate.

3.1.3 The n/RTT_{eff} parameter

After giving methods to predict the number of flows in the same RTT case and more importantly, n/RTT_{eff} when there are many TCP cross flows of different RTTs, we now see in these sections how this parameter effectively models the cross-traffic and how aggressive it is. We saw an intuitive argument for why this parameter is useful in modeling the aggressiveness of TCP cross-traffic in Section 2.2.2.

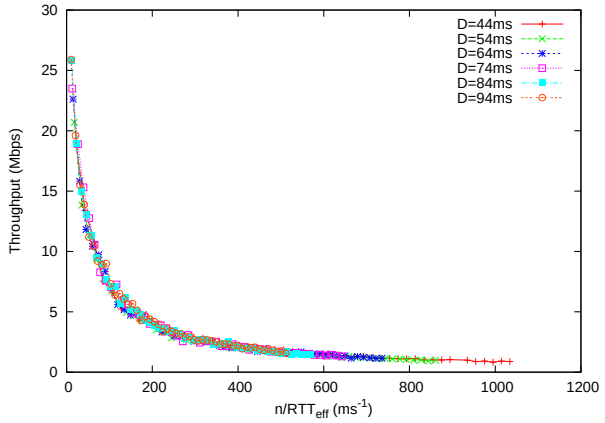

Figure 5: TCP throughput vs n/RTT_{eff}

In this experiment, we set the propagation delay of our path to 52 ms and vary the propagation delay of all other paths from 44 ms to 94 ms and measure the throughput and loss rate of our flow. Figure 5 shows the variation of TCP throughput with n/RTT_{eff}. Each of the curves shows the variation of throughput with this parameter for different values of n (number of cross flows) for given fixedf propagation delay (D) of the cross flows. We notice that for a particular value of n/RTT_{eff} (say k), the TCP throughput for our flow is similar for different combinations of n' and RTT'_{eff} (or D) that also yield $n'/RTT'_{eff} = k$. Another observation is that the TCP throughput curve is hyperbolic which can be seen by substituting $p \propto n^2$ in Equation 2. Figure 6 shows a similar graph of how loss rate varies with the above defined parameter. We not only see strikingly similar loss rates for different n's and RTTs that have the same value of the parameter, but also the quadratic behavior of loss rate versus n ($p \propto n^2$).

3.2 Emulab

We are facing a number of implementation challenges with Emulab. Currently, our setup is at a stage where we have been able to run preliminary experiments on Emulab. However:

- Changing queue sizes on Emulab has proven to be difficult, so we are currently using the default queue size of 50 slots, which is very low and restricts our link latencies to very low values. The maximum supported queue size of 100 slots is also low. Emulab does not support the `queue-limit` method of adjusting queue sizes.

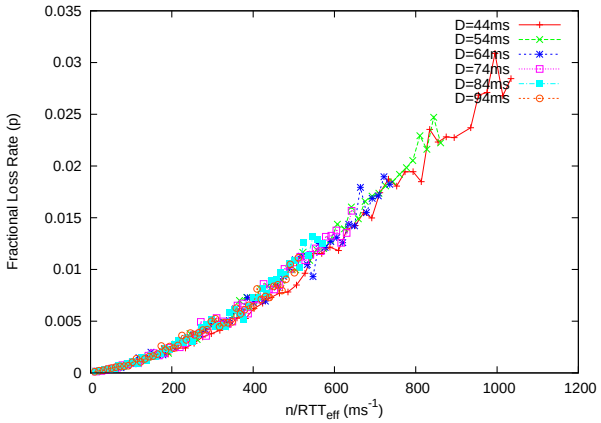

Figure 6: TCP loss rate vs n/RTT_{eff}

- For simple tasks like verifying the absence of timeouts, measuring packet loss event rates, it should be sufficient to parse `netstat` output. However, `netstat` on Emulab machines inaccurately reports a number of important statistics, such as FastRetransmits and Timeout.
- When cross-traffic is low and achieved bandwidth is high, `tcpdump` drops a significant number of packets.

Preliminary observations from Emulab:

- When multiple flows of the same RTT are present, bandwidth is only approximately fairly allocated.
- Unlike in `ns-2` where every packet is ACKed, in Emulab, only every other packet gets ACKed.
- Unlike in `ns-2`, in Emulab packets can arrive out-of-order.
- Per-flow throughput seems to vary over the duration of an experimental run.

4. RELATED WORK

PlanetLab [2], Emulab [11], and Flexlab [9], all are different approaches to achieve the same goal as us – a testbed environment for evaluating new distributed applications. As discussed previously, each approach has its unique advantages and disadvantages. PlanetLab, being an overlay testbed, exposes applications to realistic network conditions. However, those conditions are not replayable; experimenters do not have root access to the nodes they use, and in practice, nodes are frequently overloaded. Emulab addresses all of these limitations, but does so by trading off reliability. Flexlab attempts to find a middle ground. Using measurements

of PlanetLab network conditions, it allows the experimenter to construct one of various possible path models (simple-static, simple-dynamic or ACIM, in increasing order of sophistication). These path models can then be used to shape traffic generated by the experimenter's application, which is run on Emulab.

Flexlab's accuracy and repeatability seem to be at odds with each other. The most sophisticated path model, ACIM, accurately predicts the network's response to workload offered by an application. However, it is tied to the specific details of that particular application. This impacts repeatability – if a later version of the application generates a different traffic pattern, the previous ACIM model will no longer be applicable, and Flexlab will have to take new measurements of network conditions on PlanetLab.

Our objective is to develop an accurate path model that is not tied to an individual application. A fully general path model would be able to model arbitrary cross-traffic. In this work, we focus on certain restricted classes of cross-traffic, e.g. long-running timeout-free TCP flows in steady state. This allows us to pick up on established work on TCP modeling [6].

Padhye et al [6] relate bandwidth achieved by a single TCP flow to its observed packet loss event rate. Our path models are also based on observed packet loss rate, but additionally need to consider conditions where multiple flows are present (of which we are one). [5], and more recently [8], study the behavior of multiple competing TCP flows. While they do study the behavior of individual connections, they also focus on aggregate properties of the network, and their end goal is not to build a path model for an Emulab-like system. [8] analyzes variation in packet loss rates with number of competing flows, and observes that TCP fairness degrades as the number of competing flows increases. [5] studies various aggregate and per-connection metrics of TCP performance when a number of competing flows are present. They are particularly interested in very large numbers of concurrent connections (multiple hundreds).

5. CONCLUSION AND FUTURE WORK

Starting with the simple case of predicting the number of flows when the RTTs of all competing flows are equal, we gave a model-based method for inferring cross-traffic characteristics using the parameter $n/RTTeff$. We also gave simulation results of how this parameter effectively models the cross-traffic behavior and its impact on path characteristics.

One caveat about the practicality of our approach is that we have assumed that there are no short-lived flows like Web traffic and UDP traffic. Also, we have assumed that all flows have low loss rates (and no timeouts) and that these flows are all in steady state (past the slow start phase). Extending our model to incorporate some of these features would be an interesting part of future work. We also need to make our prediction mechanism more robust to minor fluctuations in aggregate bandwidth achieved and loss rates. Last but not least, we have to validate our techniques using measurement and emulation.

6. REFERENCES

[1] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. *SIGOPS Oper. Syst. Rev.*, 35(5):131–145, 2001.

[2] M. E. Fiuczynski. Planetlab: overview, history, and future directions. *SIGOPS Oper. Syst. Rev.*, 40(1):6–10, 2006.

[3] S. Floyd and E. Kohler. Internet research needs better models. *SIGCOMM Comput. Commun. Rev.*, 33(1):29–34, 2003.

[4] S. Floyd and V. Paxson. Difficulties in simulating the internet. *IEEE/ACM Trans. Netw.*, 9(4):392–403, 2001.

[5] R. Morris. Tcp behavior with many flows. In *ICNP '97: Proceedings of the 1997 International Conference on Network Protocols (ICNP '97)*, page 205, Washington, DC, USA, 1997. IEEE Computer Society.

[6] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling tcp throughput: a simple model and its empirical validation. *SIGCOMM Comput. Commun. Rev.*, 28(4):303–314, 1998.

[7] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, 2003.

[8] L. Qiu, Y. Zhang, and S. Keshav. Understanding the performance of many tcp flows. *Comput. Netw.*, 37(3-4):277–306, 2001.

[9] R. Ricci, J. Duerig, P. Sanaga, D. Gebhardt, M. Hibler, K. Atkinson, J. Zhang, S. Kasera, and J. Lepreau. The Flexlab approach to realistic evaluation of networked systems. In *Proc. of the Fourth Symposium on Networked Systems Design and Implementation (NSDI 2007)*, Cambridge, MA, Apr. 2007.

[10] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, 2002.

[11] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *OSDI '02: Proceedings of the 5th*

symposium on Operating systems design and implementation, pages 255–270, New York, NY, USA, 2002. ACM.

[12] K. Yocum, K. Walsh, A. Vahdat, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. *SIGCOMM Comput. Commun. Rev.*, 32(3):28–28, 2002.