SHRiNK: Enabling Scaleable Performance Prediction and Efficient Simulation of Networks

Rong Pan†, Konstantinos Psounis†, Balaji Prabhakar†, Damon Wischik‡
† Stanford Univerity, ‡ Cambridge University
Email: rong, kpsounis, balaji@stanford.edu, D.J.Wischik@statslab.cam.ac.uk

***Abstract*— As the Internet grows, it is becoming increasingly difficult to collect performance measurements of a network or a web-server farm, to monitor its state, and to perform simulations efficiently. Besides, the heterogeneity of the Internet makes it time-consuming and difficult to devise traffic models and analytic tools which would allow us to work with summary statistics.**

We explore a method to side-step these problems by combining sampling, modeling and simulation. Our hypothesis is this: if we take a sample of the input traffic, and feed it into a suitably scaled version of the system, we can extrapolate from the performance of the scaled system to that of the original.

Our main findings are: When we scale an IP network which is shared by short- and long-lived TCP-like and UDP flows; and which is controlled by a variety of active queue management schemes, then performance measures such as queueing delay and drop probability are left virtually unchanged. We show this in theory and in simulations. This makes it possible to capture the performance of large networks quite faithfully using smaller scale replicas.

I. Introduction

Measuring the performance of the Internet and predicting its behavior under novel protocols and architectures are important research problems. These problems are made difficult by the sheer size and heterogeneity of the Internet: it is very hard to simulate large networks and to pinpoint aspects of algorithms and protocols relevant to their behavior. This has prompted work on traffic sampling [6], [7]. Sampling certainly reduces the volume of data, but it can be hard to work backwards—to infer the performance of the original system.

A direct way to measure and predict performance is with exhaustive simulation: If we record the primitive inputs to the system, such as session arrival times and flow types, we can in principle compute the full state of the system. Further, through simulation we can test the behavior of the network under new protocols and architectures. But such large-scale simulation requires massive computing power.

Reduced-order models can go some way in reducing the burden of simulation. In some cases [12], [29] one can reduce the dimensionality of the data, for example by working with traffic matrices rather than full traces, while retaining enough information to estimate the state of the network. The trouble is that this requires careful traffic characterization and model-building. The heterogeneity of the Internet makes this time-consuming and difficult, since each scenario might potentially require a different model.

In this paper, we explore a way to reduce the computational requirements of simulations and the cost of experiments, and hence simplify network measurement and performance prediction. We do this by combining simulations with sampling and analysis. Our basic hypothesis, which we call SHRiNK (Small-scale Hi-fidelity Reproduction of Network Kinetics), is this: if we take a *sample* of the traffic, and feed it into a *suitably scaled* version of the system, we can *extrapolate* from the performance of the scaled system to that of the original.

This has two benefits. First, by relying only on a sample of the traffic, SHRiNK reduces the amount of data we need to work with. Second, by using samples of actual traffic, it short-cuts the traffic characterization and model-building process while ensuring the relevance of the results.

This approach also presents challenges. At first sight, it appears optimistic. Might not the behavior of a large network with many users and higher link speeds be intrinsically different from that of a smaller network? Somewhat surprisingly, we find that in several essential ways, one can mimic a large network using a suitably scaled-down version. The key is to find suitable ways to scale down the network and extrapolate performance.

Our main results are: (i) For networks in which flows arrive at random times and whose sizes are heavy-tailed, performance measures such as the distribution of the number of active flows and of their normalized transfer times are left virtually unchanged in the scaled system. In Section II we verify this using a simple but powerful theoretical argument. This argument reveals that the method we suggest for "SHRiNKing" networks in which flows arrive at random times will be widely applicable (i.e. for a variety of topologies, flow transfer protocols, and queue management schemes). These networks are representative of the Internet. (ii) For networks which carry long-lived TCP-like flows arriving in clusters, and which are controlled by a variety of active queue management schemes, we find a different scaling from that in Section II which leaves the queueing delay and drop probability unchanged as a function of time. In Section III we verify this using the differential-equation type models developed in [20]. (Such models have been widely used in designing control algorithms and for conducting control-theoretic analyses of network behavior.) These networks are widely used in simulations, e.g. [17], [18], [28]. (iii) Finally, we apply SHRiNK to web server farms. Experimental results with multiple machines reveal that a number of

performance metrics remain virtually unchanged.

A motivating example: Before continuing, we consider a simple example which illustrates the key points: the $M/M/1$ queue. Suppose jobs arrive at a queue according to a Poisson process of rate λ, and that service times are independent and exponential with rate $\mu > \lambda$. Let $Q(t)$ be the number of jobs in the system at time t.

Now scale the system as follows: Sample the arriving jobs, keeping each job with probability α, independent of the others, so that the sampled arrivals form a Poisson process of rate $\alpha\lambda$. Consider feeding the sampled arrivals to a separate queue whose server runs slower than the first by a factor α. This is equivalent to multiplying the service times by a factor $1/\alpha$ (so that they are rate $\alpha\mu$ exponentials), and the second queue is also $M/M/1$. If $\tilde{Q}(t)$ is the number of jobs in the slower queue at time t, then it is not hard to see that $\tilde{Q}(t) = Q(\alpha t)$ *in distribution*. That is, the evolution of the slower queue is statistically equivalent to that of the original queue slowed down in time by a factor α. This is because the queue-size process in an M/M/1 queue is a birth-death chain. The birth and death rates in the original queue are λ and μ respectively; while they are $\alpha\lambda$ and $\alpha\mu$ in the slower queue.

As a consequence, in equilibrium, the marginal distributions of the two queues are equal: i.e. $P(Q \geq n) = (\lambda/\mu)^n = (\alpha\lambda/\alpha\mu)^n = P(\tilde{Q} \geq n)$. Thus, we have inferred the distribution of queue-size, and hence of delay, in the original high-speed system by looking at a smaller-scale version.

It is natural to be skeptical of the relevance of these results. After all, they assume Poisson input traffic, whereas Internet packet traffic exhibits long-range dependence. Even more, these are open networks (the rate of arrivals is independent of current network congestion), quite different from the window flow-controlled Internet.

Nevertheless we find in the coming sections that the SHRiNK approach can be applied to IP networks, because it relies on factors other than packet level statistics: we shall see that it relies on certain fundamental scalability properties of networks.

II. IP NETWORKS WITH SHORT AND LONG FLOWS

It has been shown that the size distribution of flows on the Internet is heavy-tailed [30]. Hence, Internet traffic consists of a large fraction of short flows, and a small fraction of long flows that carry most of the traffic. Also, it has been recently argued that since network sessions arrive as a Poisson process [9], [22], [25],[1] network flows are *as if* they were Poisson [15]. (In particular, the equilibrium distribution of the number of flows in progress at any time can be obtained by assuming that flows arrive as a Poisson process.) We take these observations into account and study the scaling behavior of IP networks carrying heavy-tail distributed, Poisson flows. Such networks are a plausible representation of today's Internet.

[1]That network sessions are Poisson is not surprising since a Poisson process is known to result from the superposition of a large number of independent user processes.

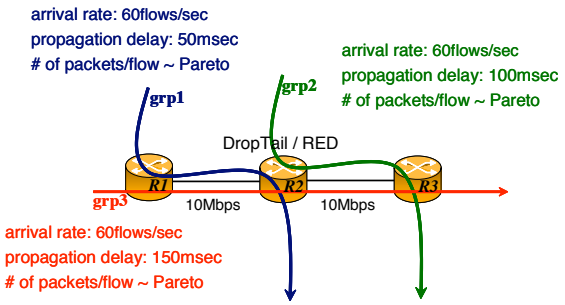

Fig. 1. Basic network topology and flow information.

A. Sampling and Scaling

We start with sampling: Due to the tremendous increase in the volume and speed of network traffic, it is very expensive to sample packets. At the other end of the spectrum, one may sample network sessions. Here is an exmaple of a network session: an end user is browing the web to download pictures; a network session starts when he/she starts web browsing and terminates when he/she stops. During the session, a new network flow starts when a particular picture starts being downloaded and terminates when the download is complete. Clearly, sampling sessions is hard in practice, because only end users have enough information to distinguish between different sessions. Hence, we choose to sample network flows.[2] This reduces the traffic we have to deal with, and is easy to implement in practice.

A second issue related to sampling is: How are the network flows sampled? The method samples the flows, choosing each one with probability α, all choices being independent. The last issue with sampling is: Where are flows sampled? The method samples at network entry points, e.g. at edge routers.

What is left is to describe how to obtain the small replica of the original network. This is done as follows: (i) link capacities are reduced by a factor α, (ii) propagation delays are scaled up by a factor $1/\alpha$, and (iii) protocol timeouts are also scaled up by the same factor. Intuitively, these steps aim to slow down the speed of the network. This will become more clear in Section II-C.

B. Simulation Results

In this section we investigate how accurately SHRiNK can predict the performance of IP networks from small-scale replicas, using the network simulator ns [21].

For simplicity, consider the topology illustrated in Figure 1. There are three routers, $R1$, $R2$ and $R3$, two links in tandem, and three groups of flows, $grp1$, $grp2$, and $grp3$. The link speeds are 10Mbps.

Routers use either the Random Early Detection (RED) or the DropTail queue management schemes. The RED parameters

[2]Notice that in accordance with the usual practice [8], [13], [14], packets are said to belong to the same flow if they have the same source and destination IP address, and source and destination port number. A flow is "on" if its packets arrive more frequently than a timeout of some seconds. This timeout is usually set to something less than 60 seconds.

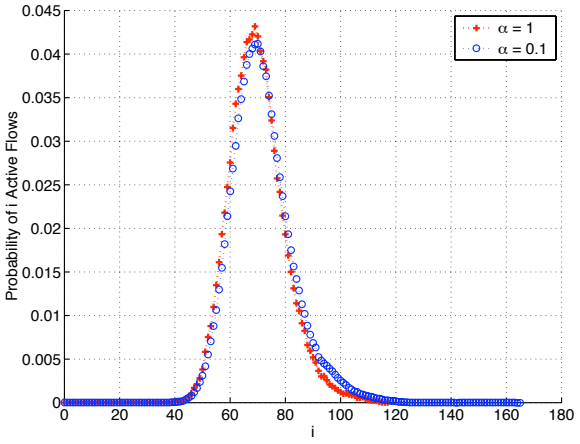

Fig. 2. Distribution of number of active flows on the first link (no drops).

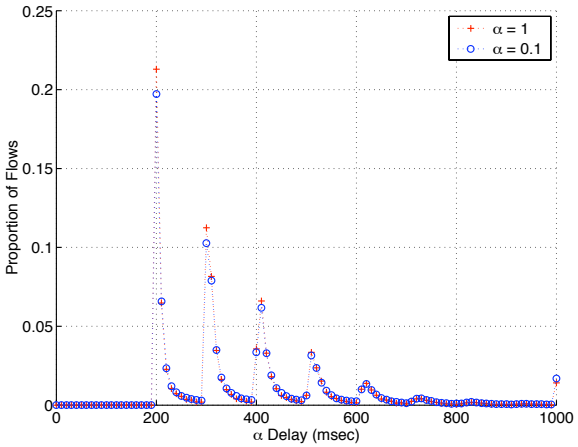

Fig. 3. Histogram of normalized delays of $grp1$ flows (no drops).

are $min_{th} = 100$, $max_{th} = 250$ and $w = 0.00005$. When using DropTail, the buffer can hold 200 packets.

Within each group, flows arrive as a Poisson process with rate λ. We vary λ to study both uncongested and congested scenarios. (We use the ns built-in routines to generate web sessions consisting of a single object each. This is what we call a flow in the simulations.) Each flow consists of a Pareto-distributed number of packets with average size 12 packets and shape parameter equal to 1.2. The packet size is set to 1000 bytes. The propagation delay of each flow of $grp1$, $grp2$, and $grp3$, is 50msec, 100msec, and 150msec respectively.

We run the experiments for scale factors $\alpha = 1$ and 0.1, and compare the distribution of the number of active flows as well as the histogram of the normalized delays of the flows in the original and the scaled system. (The normalized delays are the flow transfer times multiplied by α.) We also compare more detailed performance measures such as the distribution of active flows that are less than some size and belong to a particular group, and the distribution of the packet buffer occupancies. As will be shown in Section II-C, the method can predict the marginal and joint distributions of a large number of performance measures.

We start with the simple case where no drops occur. The flow arrival rate is set to 45 flows/sec within each group.

Figure 2 plots the distribution of the number of active flows in the first link. The two distributions match. A similar conclusion is obtained at the second link.

Figure 3 plots the histogram of the normalized delays of the flows of $grp1$. To generate the histogram, we use normalized delay chunks of 10msec each. There are 100 such delay chunks in the plot, corresponding to flows having a normalized delay of 0 to 10msec, 10msec to 20msec, and so on. The last delay chunk is for flows that have a normalized delay of at least 1sec. The plot reveals that the distribution of the normalized delays match. The results for the other two groups of flows are similar.

The peaks in the delay plot are due to the TCP slow-start mechanism. The left-most peak corresponds to flows which send only one packet and face no congestion. These flows only have to wait for the setup of the TCP connection. (Hence, for example, in Figure 3 where propagation delays are 50msec, the normalized delay for these flows is a bit more than 200msec accounting for SYN, SYN-ACK, the data packet, the ACK for the packet, and insignificant transmission and queueing delays.) The portion of the curve between the first and second peaks corresponds to flows which send only one packet and face congestion (but no drops). The next peak corresponds to flows which send two or three packets and face no congestion. These flows have to wait for an additional round trip time for the acknowledgment for the first packet to arrive. The third peak corresponds to flows which send between four and seven packets and face no congestion, and so on.[3]

We now present results for the more challenging and realistic case of congested networks. Accordingly, to induce congestion, flow arrival rates are set to 60 flows/sec within each group. Flows experience drops that account for up to 5% of the total traffic. We first present simulations where all three routers use RED.

Figure 4 plots the distribution of the number of active flows in the first and the second link. The two distributions match in both links.

Figure 5 plots the histogram of the normalized delays of the flows of $grp1$ and $grp2$. Notice that we use 150 and 200 delay chunks for the $grp1$ and $grp2$ flows respectively. Figure 6 plots the histogram of the normalized delays of the flows of $grp3$. 300 delay chunks are used in this plot. In all three cases, the delay histograms match.

What about more detailed performance measures? As an example, we compare the distribution of active flows belonging to $grp3$ that are less than 12 packets long. Figure 7 compares the two distributions from the original and scaled system. Again, the plots match.

We will now investigate if distributions scale when DropTail is used. Figure 8 plots the distribution of the number of concurrently active flows in the second link between routers $R2$ and

[3]Recall: Whenever an acknowledgment arrives, TCP senders double their window sizes.

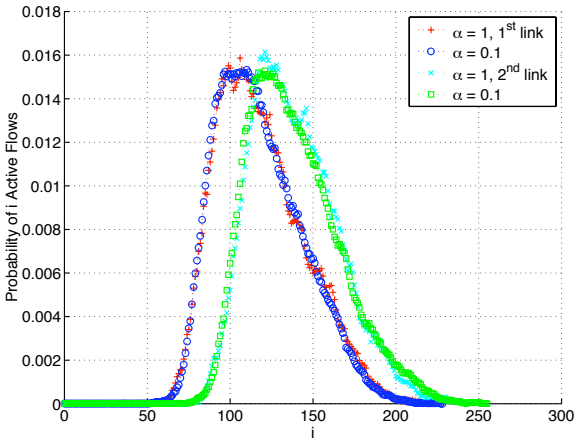

Fig. 4. Distribution of number of active flows on the first and second link (RED).

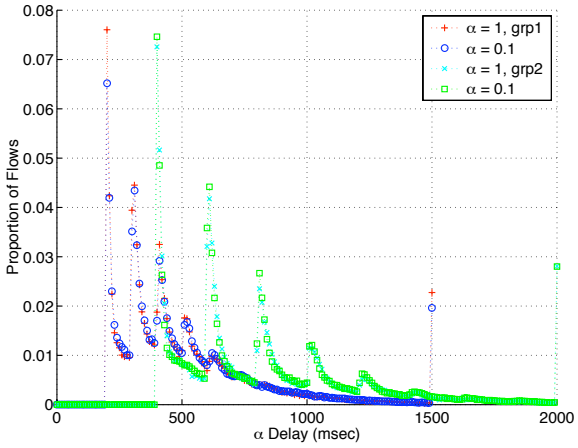

Fig. 5. Histogram of normalized delays of $grp1$ and $grp2$ flows (RED).

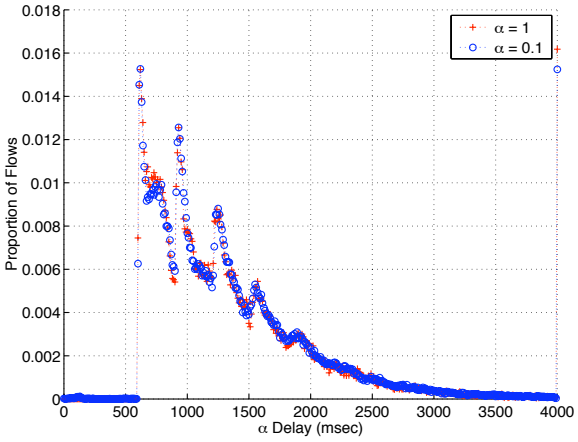

Fig. 6. Histogram of normalized delays of $grp3$ flows (RED).

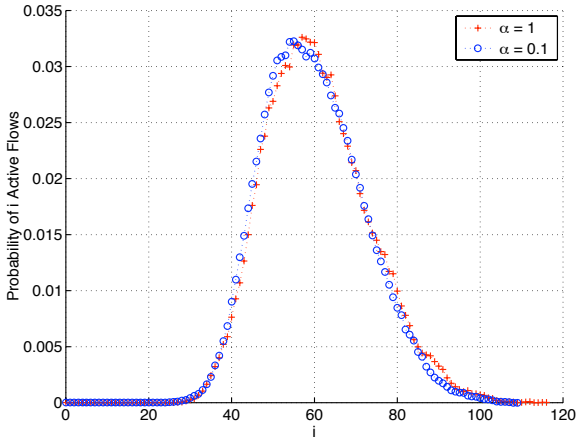

Fig. 7. Distribution of number of active $grp3$ flows with size less than 12 packets (RED).

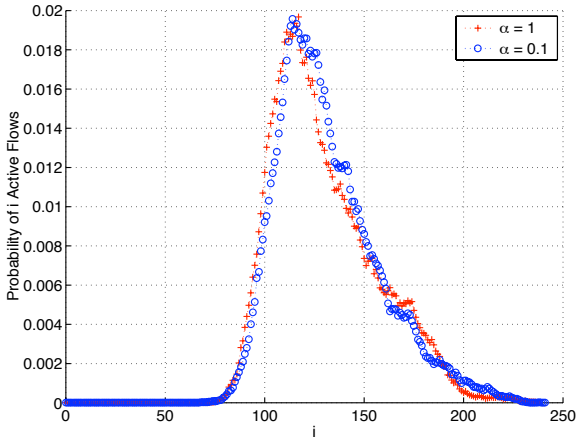

Fig. 8. Distribution of number of active flows on the second link (DropTail).

$R3$ when all routers use DropTail. It is evident from the plot that the two distributions match as before. A similar scaling holds for the other link.

Figure 9 plots the histogram of the normalized delays of the flows of $grp2$ when DropTail is employed. The distributions match as before. A similar scaling holds for the other two groups of flows.

So far, the method has successfully predicted the distribution of various performance measures at the *flow* level. Figure 10 compares the distribution of the number of *packets* at the first queue, which uses RED, in the original and scaled network. As evident from the plot, the method can also predict the distribution of the queue occupancies.

C. Understanding the Results

Recall that flows arrive as a Poisson process, bearing sizes drawn independently from a common (Pareto) distribution. [4]

[4]Note that whereas flow sizes are independent, their delays (equal to their total transfer times) are usually dependent.

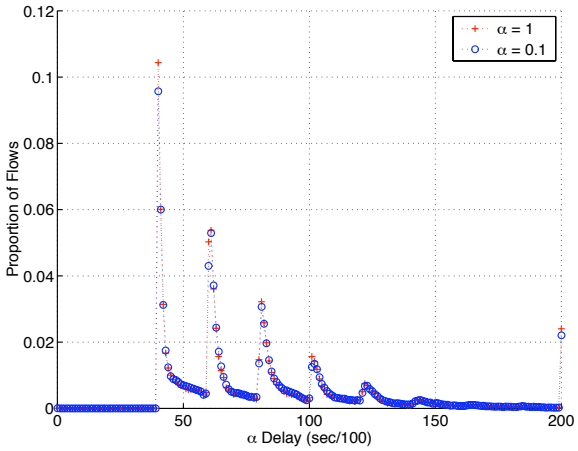

Fig. 9. Histogram of normalized delays of $grp2$ flows (DropTail).

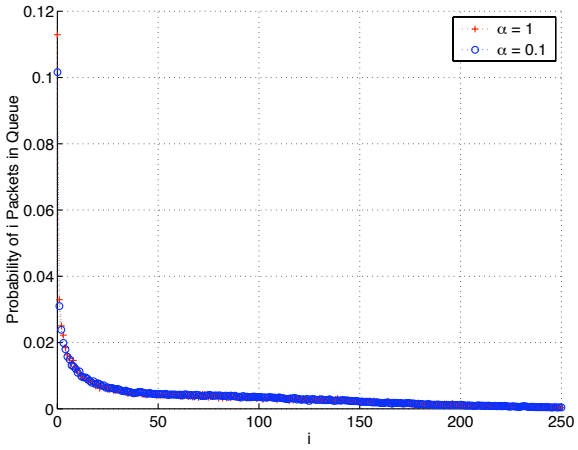

Fig. 10. Distribution of number of packets in $R1$.

By *the state of the network at time* t we mean the total information that is needed to resume the evolution of the network from time t onwards, given input data (flow arrival times and sizes) after time t. For example, the state consists of information about currently active flows: their transfered packets, their packets in transit, and where they are at time t, etc. Write $S(t)$ for the state at time t. If $I(t)$ denotes the input data to the system then $S(t)$ is some function, $\mathcal{F}$, of the input until time t. Symbolically, $S(t) = \mathcal{F}[I(s), s < t]$. We shall abbreviate this to $S(t) = \mathcal{F}[I(\cdot)]$. Note that $\mathcal{F}$ is some complicated function depending on transport protocols, queue management schemes, and other network- and user-specific details.

Theorem 1: Consider a network where network flows arrive as a Poission process bearing sizes drawn indepedently from an arbitrary distribution. Let $S(t)$ be the state of the original network at time t, and $\tilde{S}(t)$ be the state of the scaled network at time t. Then $S(\alpha t) \stackrel{d}{=} \tilde{S}(t)$, i.e. the same in distribution.

Proof: Let $I(\cdot)$ and $\tilde{I}(\cdot)$ be the inputs to the original and scaled systems, respectively. Let $\mathcal{F}^o$ and $\mathcal{F}^s$ denote the functions corresponding to the original and scaled (slowed-down)

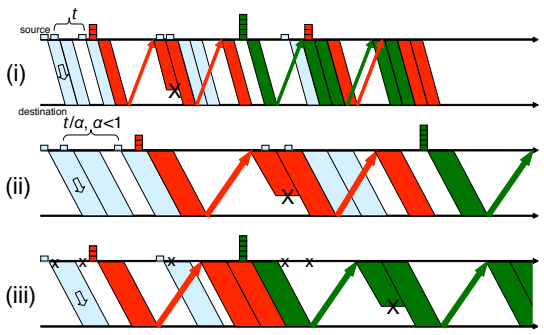

Fig. 11. Time evolution of: (i) the original, (ii) the time-stretched, and (iii) the scaled system.

networks. This gives $S(t) = \mathcal{F}^o[I(\cdot)]$ and $\tilde{S}(t) = \mathcal{F}^s[\tilde{I}(\cdot)]$. Our method of proof consists of constructing a third system, the "time-stretched system", which is obtained by applying the input $\hat{I}(t) \equiv I(\alpha t)$ to the scaled system. Thus, the time-stretched system has as input the input of the original system stretched out in time by a factor α. That is, flow f, of size s, arrives to the original system at time t iff it arrives, again with size s, to the time-stretched system at time t/α. It is a simple, but far-reaching, property of the Poisson process that $\tilde{I}(\cdot) \stackrel{d}{=} \hat{I}(\cdot)$, since sampling a proportion α of the points of a rate λ Poisson process will yield a rate $\alpha\lambda$ Poisson process. And the independent nature of the sampling process does not destroy the i.i.d. nature of the flow sizes.

Let $\hat{S}(t) = \mathcal{F}^s[\hat{I}(\cdot)]$ denote the state of the time-stretched system at time t. We shall show that the following identity is satisfied at every time t:

$$\hat{S}(t) = S(\alpha t). \tag{1}$$

Establishing this will complete our proof since $S(\alpha t) = \hat{S}(t) = \mathcal{F}^s[\hat{I}(\cdot)] \stackrel{d}{=} \mathcal{F}^s[\tilde{I}(\cdot)] = \tilde{S}(t)$.

We now establish the identity at (1). Consider the consequences of our method of scaling (slowing down) the original network: reducing link speeds by a factor α will increase queueing delays and transmission times by factor $1/\alpha$, increasing progagation delays by a factor $1/\alpha$ will increase propagation times by $1/\alpha$. Since the total delay of a packet is the sum of its queueing, transmission, and propagation times, we have effectively increased the delay of every packet by $1/\alpha$. This in turn increases the delay of every flow transfer time by a factor $1/\alpha$. It is now quite easy to see that much more is true: Since the networks are all discrete-event systems, clocked by transmissions and acknowledgments of packets, every event that occured in the original system at time t will occur in the time-stretched system at time t/α. Therefore $S(\alpha t) = \hat{S}(t)$, and the theorem is proved.

Remark 1: It is instructive to consider an illustration of the three systems, as in Figure 11. The time evolution of each of the three systems is shown between an arbitrary source-destination pair. In each sub-figure, the corresponding input process is shown on the top line. The graph of an input process denotes

flow arrival times and their corresponding sizes. The lines going upwards denote acknowledgments. Finally, the big "X"s denote packet drops. The original system has an input process of $I(t)$. For the time-stretched system, packets have larger transmission and propagation delays, denoted by "fatter" parallelograms and larger slopes respectively; and the input process, $\hat{I}(t)$, is a time-stretched version of $I(t)$. Notice that the time-stretched system is just a device for the proof, it does not exist. The input of the scaled system, $\tilde{I}(t)$, is just a subsample of the flows of $I(t)$. The unsampled flows of $I(t)$ are denoted by tiny "**x**"s on the top line of Figure 11(iii).

Remark 2: The theorem explains why the distributions of various performance measures match in distribution. Further, it shows that performance scaling involves speeding up the time, and this is why we compare normalized delays rather than delays. The proof of the theorem only relies on the assumptions about inputs (Poisson flow arrivals and i.i.d. sizes) and the fact that the network evolves as a discrete-event system. Therefore, when these assumptions are met,[5] SHRiNK is widely applicable for marginal, joint, steady-state and transient distributions of a large family of performance measures, for any network topology, transport protocol, and queue mechanism. Another consequence of Theorem 1 is that SHRiNK works for any value of α. Thus, networks can be slowed down arbitrarily. However, the smaller the α, the slower the network is, and the longer it takes for distributions to converge.

D. Applications

Since the method provides a way to deduce the performance of a fast network from a slowed-down replica, it can be used to reduce the cost of experiments: Imagine a test-network with slow network interfaces, slow switches and routers, and cheap links, that is fed with a sample of the actual network traffic.[6] In this network one may experiment with new algorithms, protocols, and architectures, and extrapolate performance.

Another use of the method is the following: There has been a recent development of research prototypes and products [5] that record partial information about the network by sampling incoming traffic. SHRiNK offers a systematic way to reproduce offline the behavior of the network using this sample.

III. IP Networks with Long-lived flows

In this section we explore how SHRiNK can be applied to IP networks used by long-lived TCP-like flows that arrive in clusters and are controlled by queue management schemes like RED. These networks are widely used to study the performance of TCP and of various AQM schemes, e.g. [17], [18], [28].

First, we explain in general terms how we sample traffic, scale the network, and extrapolate performance.

[5]We refer the reader [15] and [4] for an interesting discussion of the M/GI models and their role in generating the well-documented self-similar nature of network traffic.

[6]This network should also have larger propagation delay than the original. This can be achieved in software, or with delay-loops.

Sampling is simple. We sample a proportion α of the flows, independently and without replacement.

We scale the network as follows: link speeds and buffer sizes are multiplied by α. The various AQM-specific parameters are also scaled, as we will explain in the following section III-A. The network topology is unchanged during scaling. In the cases we study, performance measures such as average queueing delay are virtually the same in the scaled and the unscaled system.

Our main theoretical tool is the recent work on fluid models for TCP networks [20]. While [20] shows these models to be reasonably accurate in most scenarios, the range of their applicability is not yet fully understood. However, in some cases the SHRiNK hypothesis holds even when the fluid model is not accurate, as shown in Section III-A.3.

A. RED

The key features of RED are the following two equations, which together specify the drop (or marking) probability. RED maintains a moving average q_a of the instantaneous queue size q; and q_a is updated whenever a packet arrives, according to the rule

$$q_a := (1-w)q_a + wq,$$

where the w parameter determines the averaging window. The average queue size determines the drop probability p, according to the equation

$$p_{\text{RED}}(q_a) = \begin{cases} 0 & \text{if } q_a < min_{th} \\ p_{max}\left(\frac{q_a - min_{th}}{max_{th} - min_{th}}\right) & \text{if } min_{th} \le q_a < max_{th} \\ 1 & \text{if } q_a > max_{th} \end{cases} \quad (2)$$

We now explain how we scale the parameters p_{max}, min_{th}, max_{th} and w. We will multiply min_{th} and max_{th} by α. Recall that we are multiplying the buffer size by α: thus min_{th} and max_{th} are fixed to be a constant fraction of the buffer size. (This is in accord with the recommendations in [11].) We will keep p_{max} fixed at 10%, so that the drop probability is kept under 10% as long as the buffer is slightly congested. The averaging parameter w takes more thought. We shall multiply it by α^{-1}. The intuition is this: when the network is scaled down, packets arrive less frequently, so q_a is updated less often, so we make the updates larger in magnitude. Simulation and theory, described below, both indicate that this choice of scaling is natural for extrapolating performance.

1) The Basic Setup: We consider two congested links in tandem, as shown in Figure 12. There are three routers, $R1$, $R2$ and $R3$; and three groups of flows, $grp1$, $grp2$, and $grp3$. The link speeds are 100Mbps and the buffers can hold 8000 packets. The RED parameters are $min_{th} = 1000$, $max_{th} = 2500$ and $w = 0.000005$. For the flows: $grp0$ consists of 1200 TCP flows each having a propagation delay of 150ms, $grp1$ consists of 1200 TCP flows each having a propagation delay of 200ms, and $grp2$ consists of 600 TCP flows each having a propagation delay of 250ms. The flows switch on and off as shown in the timing diagram in Figure 12. Note that 75% of $grp0$ flows switch off at time 150s.

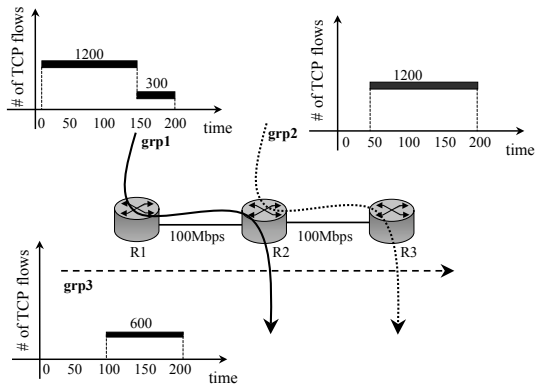

Fig. 12. Basic network topology and flow information

This network is scaled-down by factors $\alpha = 0.1$ and 0.02, and the parameters are modified as described above.

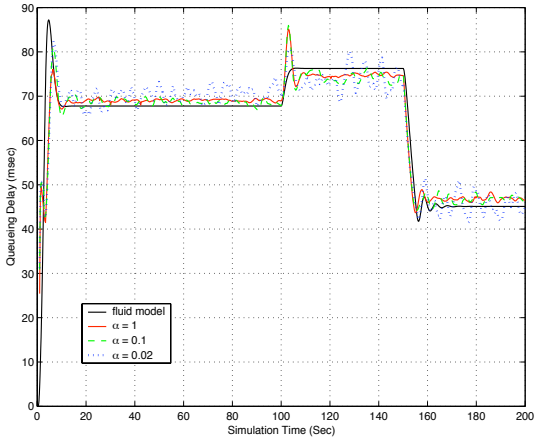

Fig. 13. Basic Setup: Average Queueing Delay at Q1

We plot the average queueing delay at Q1 and Q2 as a function of time in Figures 13 and 14. The drop probability at Q1 is shown in Figure 15. Due to limited space, we omit the plot of drop probability for Q2 since its behavior is similar to that of Q1. We see that *the queueing delay is almost identical at different scales*. (It is worth noting that it is the queueing delay which is unchanged during scaling, whereas in the $M/M/1$ model it was the queue size distribution.)

Since the drop probability is also the same in the scaled and unscaled systems, the dynamics of the TCP flows are the same. In other words, an individual flow which survives the sampling process essentially cannot tell whether it is in the scaled or unscaled system.

2) THEORY: We now show that these simulation results are supported by the recently-proposed theoretical fluid model of TCP/RED [20].

Consider N flows sharing a link of capacity C. Let $W_i(t)$ and $R_i(t)$ be the window size and round-trip time of flow i at time t. Here $R_i(t) = T_i + q(t)/C$, where T_i is the propagation

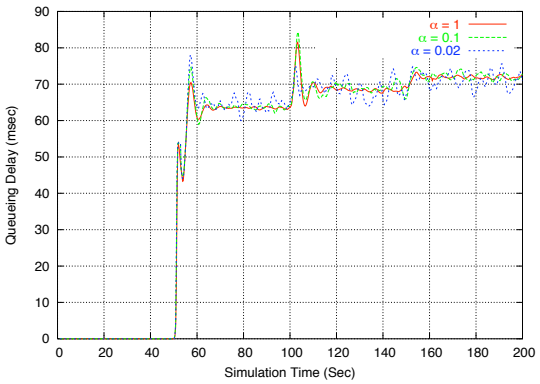

Fig. 14. Basic Setup: Average Queueing Delay at Q2

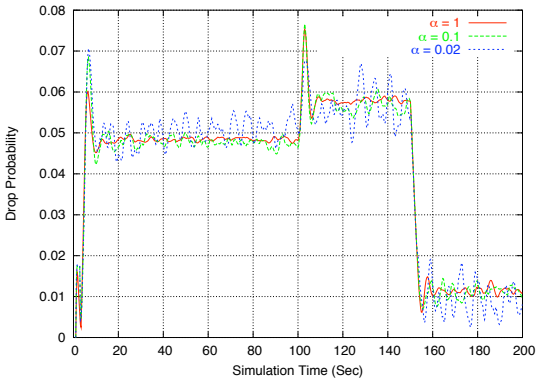

Fig. 15. Basic Setup: Drop Probability at Q1

delay and $q(t)$ is the queue size at time t. Let $p(t)$ be the drop probability at time t, and $q_a(t)$ the average queue size used by RED.

The fluid model describes how these quantities evolve; or rather, since these quantities are random, the fluid model describes how their expected values evolve. Let $\bar{X}$ be the expected value of random variable X. Then the fluid model equations are these:

$$\frac{d\bar{W}_i(t)}{dt} = \frac{1}{R_i(\bar{q}(t))} - \frac{\bar{W}_i(t)\bar{W}_i(t-\tau_i)}{1.5R_i(\bar{q}(t-\tau_i))}\bar{p}(t-\tau_i) \quad (3)$$

$$\frac{d\bar{q}(t)}{dt} = \sum_{i=1}^{N} \frac{\bar{W}_i(t)}{R_i(\bar{q}(t-\tau_i))} - C \quad (4)$$

$$\frac{d\bar{q}_a(t)}{dt} = \frac{\log(1-w)}{\delta}\bar{q}_a(t) - \frac{\log(1-w)}{\delta}\bar{q}(t) \quad (5)$$

$$\bar{p}(t) = p_{\text{RED}}(\bar{q}_a(t)) \quad (6)$$

where $\tau_i = \tau_i(t)$ solves $\tau_i(t) = R_i(\bar{q}(t-\tau_i(t)))$, δ is the average packet inter-arrival time, and p_{RED} is as in (2).

Remarks: While the applicability of these equations is not yet fully understood, [20] indicates that empirically they are reasonably accurate. Also, note that we have the constant 1.5 in (3), not 2 as in [20]. This change improves the accuracy of the fluid model for reasons elaborated in the appendix. Finally, note that while these equations describe a single link, the extension to networks is straightforward, and is given in [20].

Returning to the differential equations, suppose we have a solution to these equations

$$\big(\bar{W}_i(\cdot), \bar{q}(\cdot), \bar{q}_a(\cdot), \bar{p}(\cdot)\big).$$

Now, suppose the network is scaled and denote by C', N', etc the parameters of the scaled system. When the network is scaled, the fluid model equations change, and so the solution changes. Let $\big(\bar{W}_i'(\cdot), \bar{q}'(\cdot), \bar{q}_a'(\cdot), \bar{p}'(\cdot)\big)$ be the solution of the scaled system. We claim that, in fact,

$$\big(\bar{W}_i'(\cdot), \bar{q}'(\cdot), \bar{q}_a'(\cdot), \bar{p}'(\cdot)\big) = \big(\bar{W}_i(\cdot), \alpha\bar{q}(\cdot), \alpha\bar{q}_a(\cdot), \bar{p}(\cdot)\big).$$

If our claim is established, we will obtain that the queueing delay $\bar{q}'/C' = \alpha\bar{q}/\alpha C$ is identical to that in the unscaled system. Note also that the drop probability is the same in each case ($\bar{p}(t) = \bar{p}'(t)$). Thus, we will have theoretical support for the observations in the previous section.

Establishing the claim. We will proceed through the fluid model equations one by one. Consider first (3). Note that $R_i'(\bar{q}'(t)) = T_i + \bar{q}'/C' = T_i + \alpha\bar{q}/\alpha C = R_i(\bar{q}(t))$, so that $\tau'(t) = \tau(t)$. Hence

$$\frac{d\bar{W}_i'(t)}{dt} = \frac{1}{R_i'(\bar{q}'(t))} - \frac{\bar{W}_i'(t)\bar{W}_i'(t-\tau')}{1.5R_i'(\bar{q}'(t-\tau'))}\bar{p}'(t-\tau').$$

Consider next (4). Suppose for simplicity that all flows have identical routes. Then the W_i are statistically identical, hence the expectations $\bar{W}_i$ are all equal. So we can rewrite the equation as

$$\frac{d\bar{q}(t)}{dt} = \frac{N\bar{W}_1(t)}{R_1(\bar{q}(t-\tau'))} - C.$$

It is then easy to see that

$$\begin{aligned}\frac{d\bar{q}'(t)}{dt} &= \alpha\frac{d\bar{q}(t)}{dt} \\ &= \frac{N'\bar{W}_1'(t)}{R_1'(\bar{q}'(t-\tau'))} - C'.\end{aligned}$$

This extends to the case of multiple groups of flows with different routes, provided we sample a proportion α from each group.

Consider next (5). Recall that $w' = w/\alpha$. Note that the average packet inter-arrival time increases as the number of flows and the capacity decrease, in proportion $\delta' = \delta/\alpha$. Making the approximation $\log(1-w/\alpha) \approx \log(1-w)/\alpha$, good for small w, we see that $\log(1-w')/\delta' \approx \log(1-w)/\delta$, and hence that

$$\frac{d\bar{q}_a'(t)}{dt} \approx \frac{\log(1-w')}{\delta'}\bar{q}_a'(t) - \frac{\log(1-w')}{\delta'}\bar{q}'(t).$$

In fact, we chose $w' = w/\alpha$ so that this equation would be satisfied, allowing us to scale properly.[7]

Consider finally (6). Recall that $p'_{max} = p_{max}$, and that $min'_{th} = \alpha min_{th}$ and $max'_{th} = \alpha max_{th}$. It is then clear that

$$\bar{p}'(t) = p'_{max}\Big(\frac{\bar{q}_a'(t) - min'_{th}}{max'_{th} - min'_{th}}\Big).$$

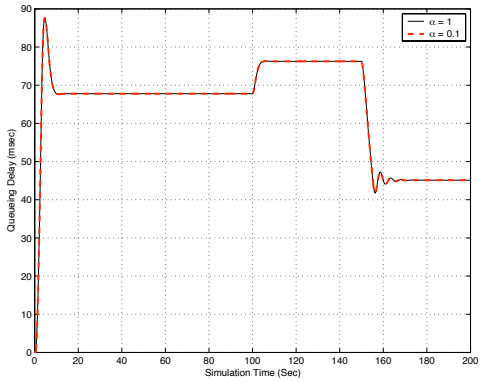

Fig. 16. Fluid model predicts scaling behavior

This establishes the claim.

Figure 16 presents the solution of the fluid model for the queueing delay at Q1 under the scenario of Figure 1 for the scale parameters $\alpha = 1$ and 0.1. As can be seen, both the solutions are virtually identical, providing a numerical illustration of the scaling property of the differential equations established above.

Remarks: It is worth remarking on a theoretical nicety related to the scaling property of these differential equations. If they had been derived from a limiting procedure in which the number of users, link capacities and buffer sizes all increase proportionally with N, then the scaling behavior would have been entirely expected (one has only to set N to equal αN before taking limits). However, they have been derived via a different route in [20]: by assuming that packet drops occur as a Poisson process. Therefore, the scaling property they exhibit is rather stunning. It strongly suggests that, in fact, they describe the behavior of the network in a large-N limit.

We also draw attention to some interesting features of all of the performance-related figures in this section. Note that transients are pretty well mimicked at the smaller scales. Also note that the smaller scale plots look more jagged, as if they are a noisy version of the original plots. The last point would be an easy consequence of a limit theorem: If in the large-N limit the behavior of the network is describable using deterministic differential equations, then away from the limit (at smaller and smaller scales) a corresponding Central Limit Theorem would suggest that the noise would be proportional to $1/\sqrt{\alpha}$.

3) WITH FASTER AND SLOWER LINKS: Suppose we alter the basic setup, by increasing the link speeds to 500Mbps, while keeping all other parameters the same. Figure 17 (zoomed in to emphasize the point) illustrates that, once again, scaling the network does not alter the queueing delay. Note that under these conditions the queue oscillates. There have been various proposals for stabilizing RED [18], [23]. We are not concerned with stabilizing RED here: we mention this case to show that SHRiNK can work whether or not the queue oscillates.

[7] It is true that w' needs to be less than 1. However, this would not be a limiting factor for the magnitude of scaling since w is generally set to a small value for high speed links: for example 10^{-6} for an 1Gbps link.

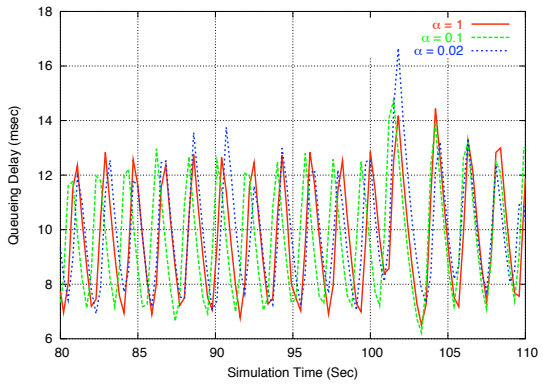

Fig. 17. With faster links: Average queueing delay at Q1 (zoomed in)

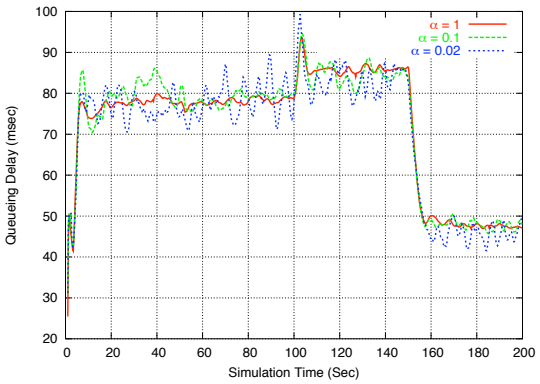

Fig. 19. With web traffic: Average Queueing Delay at Q1

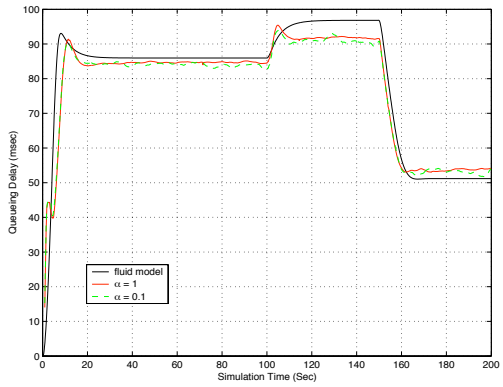

Fig. 18. With slower links: Average queueing delay at Q1

Suppose we instead alter the basic setup, by decreasing the link speeds to 50Mbps, while keeping all other parameters the same. Once again, scaling the network does not alter the queueing delay. For such a simulation scenario, especially in the time frame 100sec-150sec, the fluid model is not a good fit (see Figure 18). This is not unexpected [27]: actual window and queue sizes are integer-valued whereas fluid solutions are real-valued; rounding errors are non-negligible when window sizes are small as is the case here. The range of applicability of the fluid model is not our primary concern in this paper: we mention this case to show that SHRiNK can work whether or not the fluid model is appropriate.

4) WITH WEB TRAFFIC: So far we have only considered long-lived flows to which fluid models can be applied. We now introduce short-lived web flows to each flow group in the basic setup. Each session consists of multiple requests, each request being for a single file. The number of requests within a session is random (we use the standard ns settings), and file sizes are Pareto distributed with an average of 12 packets and a shape parameter of 1.2. In our experiment on the unscaled network, $20,000$ web sessions were generated. In the scaled version, we sample a proportion α of these sessions independently. We also sample a proportion α of the original long-lived TCP flows, as before.

Figure 19 shows that scaling the network does not affect the queueing delay much, even in the presence of web traffic. Note that here the queueing delay is dominated by the behavior of long-lived TCP flows which have reached steady state.

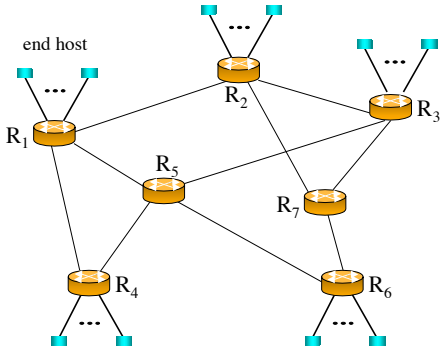

Fig. 20. A more complex topology

5) IN A MORE COMPLEX NETWORK: As a further validation, we test SHRiNK in a more complex network, shown in Figure 20. There are seven routers R1 to R7. Links R1−R2, R2−R3, R1−R5, R3−R5 and R4−R5 run at 150 Mbps, links R1−R4 and R5−R6 run at 100 Mbps, and all other links run at 50 Mbps. The traffic is a mixture of UDP and web flows; and long-lived TCP, AIMD and Binomial [2] flows. These last types have the following common form: on receiving an acknowledgement, increase the congestion window w by aw^{n-1} (TCP uses $a = 1$, $n = 0$), and on incurring a mark/drop, decrease w by bw^m (TCP uses $b = w/2$, $m = 1$). The parameters $(a, n; b, m)$ describe each class.

We omit a detailed description of all the flows, except those traversing link R5-R6 whose queueing dynamics are shown in Figure 21. Link R1→R5 carries 1000 long-lived flows, divided into five groups: 200 normal TCP, 200 AIMD $(1, 0; .1, 1)$, 200 AIMD $(2, 0; .5, 1)$, 200 Binomial $(1, 1; .5, 1)$ and 200 Binomial $(1.5, -1; .5, 1)$. The links are controlled by RED with $min_{th} = 1000$, $max_{th} = 2000$ and $w = 0.000005$. As before, we see that scaling the network does not affect the queueing delay.

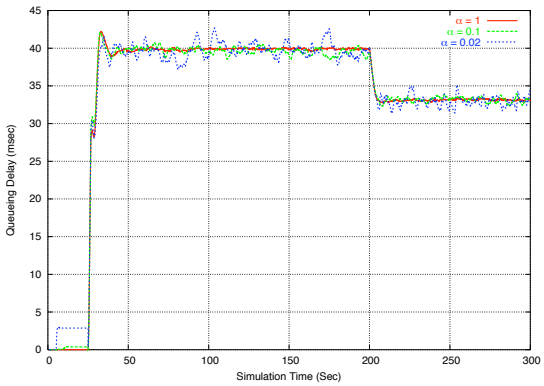

Fig. 21. In a more complex network: Average queueing delay at R5-R6

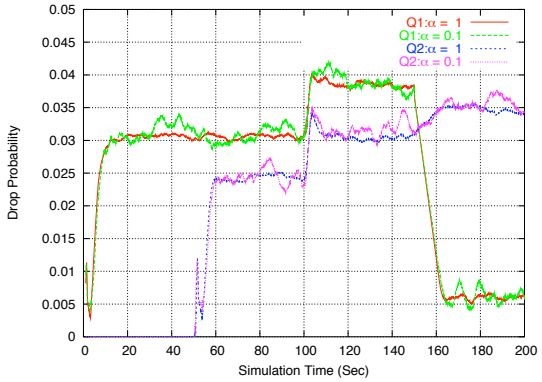

Fig. 23. PI Controller: Drop probabilities at Q1 and Q2

B. Proportional-Integral (PI) Controller

A different AQM scheme is the PI controller [17], which attempts to stabilize the queue size around a given target value. The PI controller drops/marks packets with a probability p which is updated periodically by

$$p(t+\delta t) = p(t) + a\big(q(t+\delta t) - q_{\text{target}}\big) - b\big(q(t) - q_{\text{target}}\big). \quad (7)$$

Here, q is the instantaneous queue size, q_{target} is the target queue size, δt is the update timestep (here fixed at 0.01s), and a and b are arbitrary parameters.

We first explain how we will scale the network. As usual, let a' etc. be the scaled parameters. We will sample a fraction α of the flows, and set $a' = a/\alpha$, $b' = b/\alpha$ and $q'_{\text{target}} = \alpha q_{\text{target}}$. (This is in accordance with the design rules in [17].)

We simulated the basic setup of Section III-A, replacing RED by the PI controller. We use $a = 8.8681 \times 10^{-7}$ and $b = 8.7427 \times 10^{-7}$, as suggested in [17]. We set q_{target} to be 1750 packets, which is half-way between our min_{th} and max_{th} parameters from the last section.

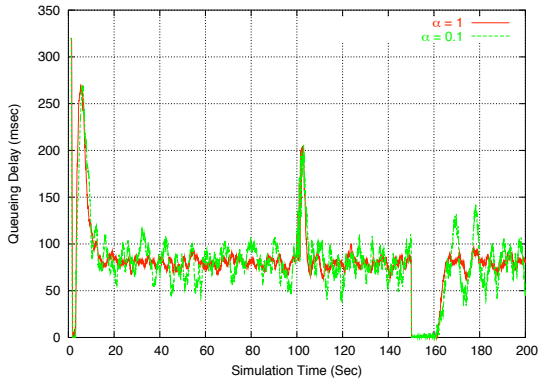

Fig. 22. PI Controller: Average queueing delay at Q1

Figure 22 shows the average queueing delay at different scales for Q1. We see that scaling the network does not affect queueing delay, at least in steady state. There are some spikes when the load changes abruptly, and the small-scale network shows slightly larger spikes. Figure 23 shows that neither is the drop probability affected by scaling the network.

We can again use the fluid model to understand this behaviour. To obtain the fluid model for the PI controller, we simply replace (5) and (6) in the fluid model by the fluid analog of (7): the expected drop probability $\bar{p}$ evolves according to

$$\frac{d\bar{p}}{dt} = -b\frac{d\bar{q}}{dt} + (b-a)\big(\bar{q}(t) - q_{\text{target}}\big).$$

As before, by our choice of scaling,

$$\frac{d\bar{p}}{dt} = \frac{d\bar{p}'}{dt} = -b'\frac{d\bar{q}'}{dt} + (b'-a')\big(\bar{q}'(t) - q'_{\text{target}}\big).$$

Thus the fluid model also scales.

C. AVQ

Another type of active queue management scheme is the adaptive virtual queue (AVQ) [19], an extension of the virtual queue algorithm [16]. The idea of AVQ is to adapt the marking probability to reach some given target utilization. It does this by running a virtual queue in parallel with the actual queue, and marking packets which arrive when the virtual queue is full.

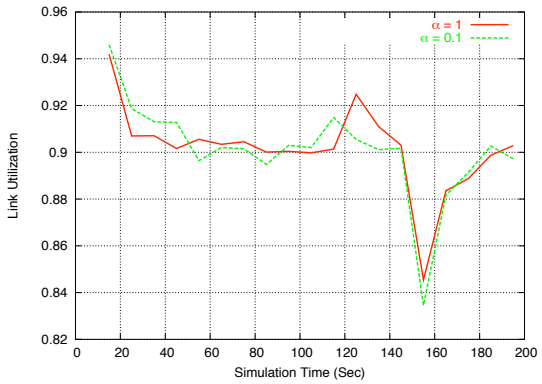

Fig. 24. AVQ: Link Utilization

The easiest way to give more details about the algorithm is via the fluid equations suggested in [19]. Let γ be the target utilization, let C be the actual service rate of the queue, and

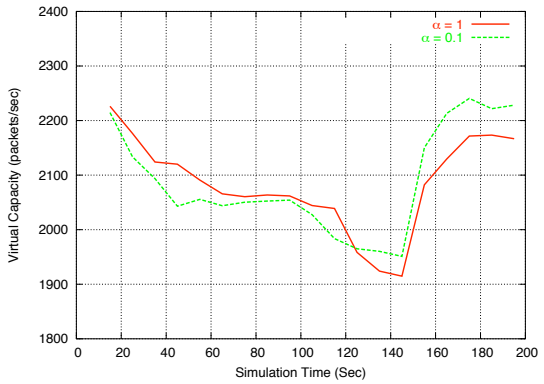

Fig. 25. AVQ: Virtual Capacity

the service rate of the virtual queue $\tilde{C}$ is dynamically adjusted according to

$$\frac{d\tilde{C}(t)}{dt} = \kappa\big(\gamma C - \lambda(t)\big) \tag{8}$$

where $\lambda(t)$ is the arrival rate at time t and κ is an arbitrary gain parameter.

How should the parameters γ and κ be scaled? Since γ is the target link utilization which is independent of any specific link speed, it is left unchanged. As a result, κ is also left unchanged in order to properly scale Equation (8).

The basic setup of Section III-A is simulated, replacing RED by AVQ. The parameters $\gamma = 90\%$ and $\kappa = 0.1$ are used at both links. Figures 24, and 25 show that scaling the network does not affect essentially the link utilization or the virtual queueing delay. Similar results hold for the marking probability.

This is also reflected in the fluid model for AVQ, which consists of Equation (8) and the following equations:

$$\frac{d\bar{W}_i(t)}{dt} = \frac{1}{T_i} - \frac{\bar{W}_i(t)\bar{W}_i(t-T_i)}{1.5T_i}\bar{p}(t-T_i) \tag{9}$$

$$\bar{p}(t) = \frac{(\lambda(t)-\tilde{C}(t))^+}{\lambda(t)} \tag{10}$$

$$\lambda(t) = \sum_{i=1}^{N} \frac{\bar{W}_i(t)}{T_i}. \tag{11}$$

The first equation is a modified version of Equation (3), modified to remove queueing delay, as AVQ should keep the (actual) buffer empty. The last two equations are from [19]. Recall that T_i is the propagation delay for user i.

Now, suppose that $\bar{W}_i(\cdot)$, $\bar{p}(\cdot)$, $\lambda(\cdot)$ and $\tilde{C}(\cdot)$ are a solution to the fluid equations. Consider the fluid equations for the scaled network. It is not difficult to check that $\bar{W}_i(\cdot)$, $\bar{p}(\cdot)$, $\alpha\lambda(\cdot)$ and $\alpha\tilde{C}(\cdot)$ solve these scaled equations.

D. DropTail

In all the examples we have studied in this section—with heterogeneous end-systems, with different of active queue management policies, and with a range of system parameters—we have found that basic performance measures such as queueing delay are left unchanged, when we sample the input traffic and scale the network parameters in proportion. This conclusion is supported by the theory of fluid models, and even holds where the fluid models fail. A notable exception is provided by the queue management scheme DropTail, as described next.

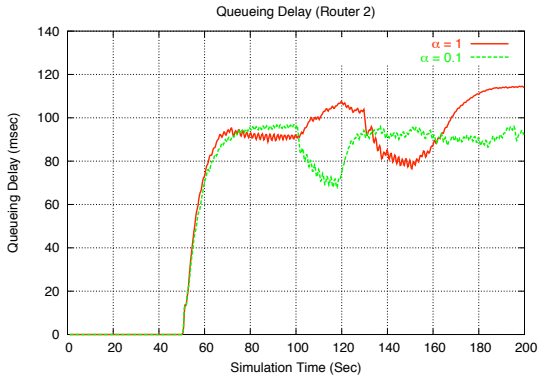

Fig. 26. DropTail: Average queueing delay at Q2

Consider the basic network setup of Section III-A, and suppose that the routers use DropTail instead of RED. Figure 26 shows the average queueing delay at Q2. Clearly, the queueing delays at different scales do not match. DropTail drops all the packets that arrive at a full buffer. As a result, it could cause a number of consecutive packets to be lost. These bursty drops underlie the failure of the scaling hypothesis in this case, as explained in [24]. Separately, note that when packet drops are bursty and correlated, the assumption that packet drops occur as a Poisson process (see [20]) is violated and the differential equations become invalid. The connection between these two phenomena (the failure of the scaling hypothesis and the invalidation of the differential equation models) is explored in [24].

E. Applications

In this section, we find that, for certain IP networks supporting flows that arrive in clusters, SHRiNK can predict the time-wise performance of a high-speed network using its properly scaled-down replica. Although in reality flows don't arrive in clusters, this type of flow arrivals has been used extensively in the design of AQM schemes and in the analysis of TCP's performance [17], [10], [18], [19], [20], [28]. Most of this work demands time-consuming ns simulations, especially for high speed links. Under these scenarios, the SHRiNK method comes as a remedy to facilitate packet-level simulations, since it drasticaly reduces the simulation time.

To illustrate the potential savings in resources, we report the CPU time to run the simulations in Section III-A.1 (RED:The Basic Setup) and Section III-A.3 (RED:Faster and Slower Links). As shown in Figure 27, the CPU time rises monotonically as α increases. The reason behind this is the fact that, for an event-driven simulator like ns, to simulate a network with more packet arrivals would mean to process more events. Naturally, one would expect that the simulation time for $\alpha = 0.5$ would be half the simulation time for $\alpha = 1$. Surprisingly, we find that the reductions of the CPU time are slightly more than half in all three cases shown in Figure 27. Generally, the slopes

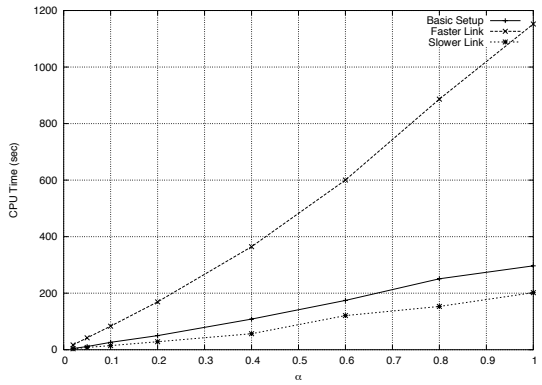

Fig. 27. CPU Time Comparison

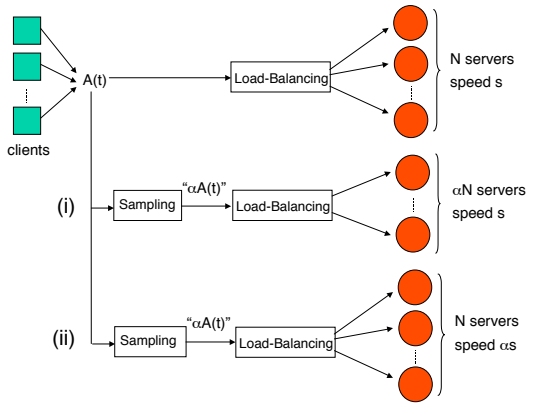

Fig. 28. Scaling a web server farm: (i) scaling the number of servers, (ii) scaling the speed of the servers.

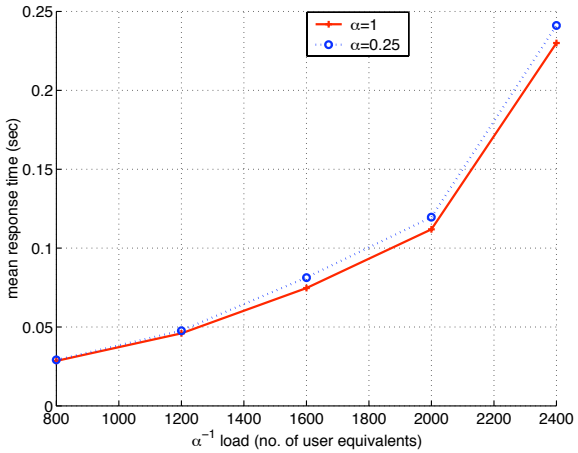

Fig. 29. Average response time, when sampling user equivalents.

of increase are greater than α^{-1}.[8] As a result, our method can shrink the network simulation time in a superlinear fashion.

IV. Web Server Farms

In this section we briefly outline how SHRiNK may apply to web server farms. Since a rapid growth in the size and capacity of web server farms makes it increasingly difficult to take performance measurements and to evaluate new algorithms and architectures, if SHRiNK applies to web server farms it would help reduce this difficulty significantly.

How should server farms be scaled? Consider a web server farm with N servers each having speed s, as in Figure 28.[9] Sample the requests for the original farm, retaining each independently with probability α. Feed the sampled traffic into a scaled-down farm consisting of either (i) a fraction α of the original web servers, or (ii) the same number of servers each having speed αs (see (i) and (ii) of Figure 28). Of interest is the closeness of the average response time, the server throughput and capacity (maximum throughput) in the scaled system to that in the original system.

We conducted some preliminary experiments using eight Linux machines configured with a Pentium III at 550MHz and 384MB of RAM, connected to a 100Mbits/sec switch. A number of the machines constitute the web farm and each hosts one Apache 1.3.9 [1] web server. The rest of the machines act as clients, each of which run Surge [3] to generate HTTP requests. We report experimental results for the case where one scales the number of servers (as illustrated in Figure 28(i)). This scaling is very useful in practice since it reduces the *size* of the system.

In the first experiment the original farm consists of four machines. The clients use HTTP1.1, load-balancing is a simple round-robin scheme, and both load-balancing and sampling take place at the user-equivalent level. (Surge uses the notion of user equivalents to generate sequences of requests similar to those generated by web sessions that stay "on" throughout the experiment.) The scaled system consists of a stand-alone server.

Figures 29 and 30 show the average response time and the normalized server throughput as a function of the normalized load. (Normalized quantities are quantities multiplied by α^{-1}.) Scaling the system leaves these quantities virtually unchanged. Note that we treat the farm of the four servers as a single entity. The normalized load is the total normalized load directed into the farm, and the normalized throughput is the sum of the normalized throughputs of the servers of the farm.

In the second experiment the original farm consists of two machines. The clients use HTTP1.0, load-balancing again is a round-robin scheme that takes place at the user-equivalent level, while sampling takes place at the HTTP request level. (We do not sample embedded requests but rather requests for whole documents.) The scaled system is a stand-alone server.

Figures 31 and 32 show the average response time and the normalized server throughput as a function of the load.[11]

[8] We believe that the extra time saving comes from machine-related issues such as memory requirements, etc. We did not investigate further.

[9] This is a simplified picture of a farm, since the application-servers, the databases, and the switches used to interconnect the various components are absent.

[11] The number of user equivalents sending requests at the two systems is now the same hence the horizontal axis is not multiplied with α^{-1} as before. It is

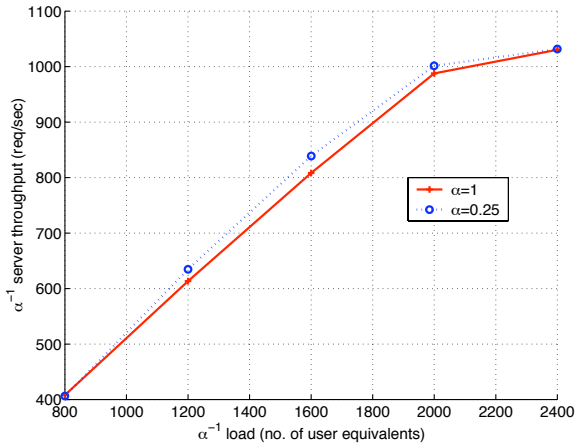

Fig. 30. Server throughput, when sampling user equivalents.

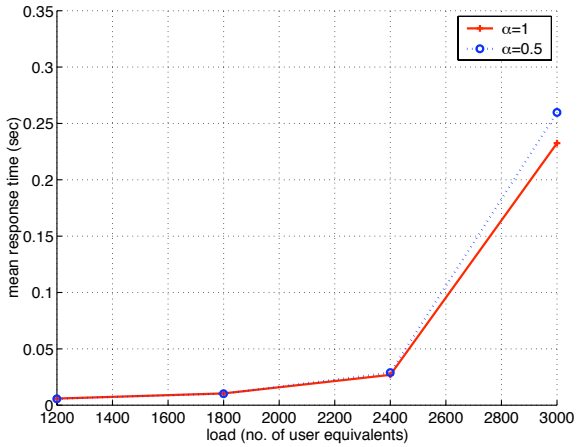

Fig. 31. Average response time, when sampling document requests.

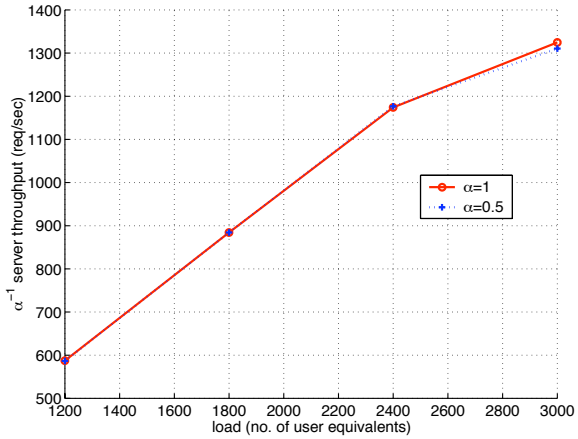

Fig. 32. Server throughput, when sampling document requests.

	Section II	Section III
flow sizes	heavy-tailed	long-lived
flow arrivals	at Poisson-like times	in clumps
physical scaling	sample flows and reduce link capacities	
	increase propagation delays	reduce buffer sizes
performance scaling	in distribution	in time
applicability	general	not valid for DropTail
usage	experimentation	simulations
the smaller the α	the longer to converge	the noisier the results

TABLE I
SHRINKING NETWORKS.

Again, these quantities remain virtually unchanged after scaling. More experimental results can be found in [26].

The results of this section are encouraging. In paricular, we have shown via experiments that a web farm consisting of a round robin load balancer and a number of web servers attached to it, can be scaled down when traffic sampling and load balancing occurs at the HTTP request or the web session level. However, it should be noted that large web farms can have complex architectures whose topological scaling might be more involved than simply scaling the number of servers.

V. CONCLUSION

In this paper we have described an approach, called SHRiNK, for scaleable performance prediction and efficient simulation of large networks.

Our first example concerned a network in which TCP flows arrive at Poisson-like times and are heavy-tailed distributed. This is a plausible representation of the Internet.[12] To construct the network replica, in addition to sampling flows and reducing link speeds, we increased propagation delays and protocol timeouts. We showed that the distribution of a large number of performance measures of the original network can be accurately predicted by the replica, irrespective of the network topology, the protocols, and the AQM schemes used. This type of scaling can be used to reduce the cost of experiments since all the hardware components will run slower. The cost to pay is time; one needs to wait longer, in real time, for the distribution of the various metrics to converge on the scaled system.

Our second example was a congested network of long-lived TCP-like flows that arrive in clusters. This is a popular network model for designing and testing new algorithms. To construct the network replica, in addition to sampling flows and reducing link speeds, we scaled down buffer sizes and AQM parameters. We showed that various performance measures can be predicted as a function of time, for a large class of networks. A notable exception is networks that use DropTail as an AQM scheme. This type of scaling can be used in simulations to reduce execution time. The cost to pay is accuracy; the smaller the scaling

the number of requests directed at the two systems that differ due to document sampling.

[12] Since Internet sessions are Poisson [9], Internet flows can be considered *as if* they were Poisson [15].

factor the more noisy the predictions are. The above points are summarized in Table I.

Finally, we have proposed a way to apply SHRiNK to web server farms. Our experimental testbed consisted of tens of machines; some generated HTTP traffic and some were organized in a web farm replying to these requests. While the application of SHRiNK to networks leaves the network topology unchanged, in the web farm case we experimented with scaling the topology too. Our results were encouraging.

REFERENCES

[1] The Apache web-server. `http://httpd.apache.org`, accessed January 2002.

[2] D. Bansal and H. Balakrishnan. Binomial congestion control algorithms. In *Proceedings of INFOCOM*, 2001.

[3] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of the ACM SIGMETRICS Conference*, June 1998.

[4] T. Bonalds, A. Prutiere, G. Gegnie, and J. Roberts. Insensitivity results in statistical bandwidth sharing. In *Proceedings of ITC17*, September, 2001.

[5] Cisco. NetFlow services and applications. White paper, 2000. `http://cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm`.

[6] Kimberly Claffy, George Polyzos, and Hans-Werner Braun. Applications of sampling methodologies to network traffic characterization. In *Proceedings of SIGCOMM*, 1993.

[7] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, 2001.

[8] W. Fang and L. Peterson. Inter-as traffic patterns and their implications. In *Proceedings of the 4th Global Internet Symposium*, December 1999.

[9] A. Feldmann, A. C. Gilbert, and W. Willinger. Data networks as cascades: Investigating the multifractal nature of internet wan traffic. In *Proceedings of ACM SIGCOMM*, 1998.

[10] P. Fernando, W. Zhikui, S. Low, and J. Doyle. A new TCP/AQM for stable operation in fast networks. In *Proceedings of INFOCOM*, 2003.

[11] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transaction on Networking*, pages 397–413, 1991.

[12] Fluid models for large, heterogeneous networks. `http://www-net.cs.umass.edu/fluid/`, accessed January 2002.

[13] C. Fraleigh, C. Diot, B. Lyles, S. Moon, P. Owezarski, D. Papagiannaki, and F. Tobagi. Design and deployment of a passive monitoring infrastructure. In *Proceedings of the Workshop on Passive and Active Measurements, PAM*, April 2001.

[14] C. Fraleigh, S. Moon, C. Diot, B. Lyles, and F. Tobagi. Packet-level traffic measurements from a tier-1 IP backbone. Technical Report TR01-ATL-110101, Sprint ATL Technical Report, November 2001.

[15] S. Ben Fredj, T. Bonalds, A. Prutiere, G. Gegnie, and J. Roberts. Statistical bandwidth sharing: a study of congestion at flow level. In *Proceedings of SIGCOMM*, 2001.

[16] R. Gibbens and F. Kelly. Distributed connection acceptance control for a connectionless network. In *Proceedings of 16th Intl. Teletraffic Congress*, 1999.

[17] C.V. Hollot, V. Misra, D. Towlsey, and W. Gong. On designing improved controllers for AQM routers supporting TCP flow. In *Proceedings of INFOCOM*, 2001.

[18] C.V. Hollot, V. Misra, D. Towsley, and W. Gong. A control theoretic analysis of RED. In *Proceedings of INFOCOM*, 2001.

[19] S. Kunniyur and R. Srikant. Analysis and design of an Adaptive Virtual Queue (AVQ) algorithm for active queue management. In *Proceedings of SIGCOMM*, 2001.

[20] V. Misra, W. Gong, and D. Towsley. A fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. In *Proceedings of SIGCOMM*, 2000.

[21] Network simulator. `http:www.isi.edu/nsnam/ns`, accessed June 2000.

[22] C. J. Nuzman, I. Saniee, W. Sweldens, and A. Weiss. A compound model for TCP connection arrivals. In *Proceedings of ITC Seminar on IP Traffic Modeling*, Monterey, 2000.

[23] T. Ott, T. Lakshman, and L. Wong. SRED: Stabilized RED. In *Proceedings of INFOCOM*, 1999.

[24] R. Pan, B. Prabhakar, K. Psounis, and M. Sharma. A study of the applicability of a scaling-hypothesis. In *Proceedings of ASCC*, 2002.

[25] Vern Paxson and Sally Floyd. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995. `citeseer.nj.nec.com/article/paxson94widearea.html`.

[26] K. Psounis. *Probabilistic Methods for Web Caching and Performance Prediction of IP Networks and Web Farms*. PhD thesis, Stanford University, December 2002. `www.stanford.edu/~kpsounis/thesis.html`.

[27] S. Shakkottai and R. Srikant. How good are deterministic fluid models of internet congestion control. In *Proceedings of Infocom*, 2002.

[28] P. Tinnakornsrisuphap and A. Makowski. Limit behavior of ECN/RED gateways under a large number of TCP flows. In *Proceedings of INFOCOM*, 2003.

[29] Jean Walrand. A transaction-level tool for predicting TCP performance and for network engineering. In *MASCOTS*, 2000. `http://walrandpc.eecs.berkeley.edu/Papers/mascots1.pdf`.

[30] Walter Willinger, Murad S. Taqqu, Robert Sherman, and Daniel V. Wilson. Self-similarity through high-variability: Statistical analysis of ethernet lan traffic at the source level. *IEEE/ACM Transactions on Networking*, 5(1):71–86, 1997.

[31] Wolff. *Stochastic Networks and the Theory of Queues*. Prentice Hall, 1989.

VI. APPENDIX

The appendix describes the derivation of the factor 1.5 in (3), the fluid equation for TCP's congestion window, and generalizes the fluid model to Binomial-TCP.

First, consider a slightly different version of that equation:

$$\frac{d\bar{W}_i(t)}{dt} = \frac{1}{R_i(\bar{q}(t))} - \frac{1}{2}\frac{\bar{W}max_i(t)\bar{W}_i(t-\tau)}{R_i(\bar{q}(t-\tau))}\bar{p}(t-\tau),$$

where $\bar{W}max$ is the typical window size just before it is cut in half. This equation reflects the fact that, when a mark is received, the window is larger than average.[13] If we can say just how much larger than average, specifically if we can explain why $\bar{W}_i = \frac{3}{4}\,\bar{W}max_i$, it will be clear that these two versions of (3) are identical.

Consider the evolution of the TCP congestion window. There are periods when it grows linearly, and instants when it is cut in half. When it is cut in half, it drops from $\bar{W}max$ to $\bar{W}max/2$. It is easy to see that the long-run average window size must be $\bar{W} = 1/2\;\bar{W}max + 1/2\;\bar{W}max/2$, giving the result.

We formalize and generalize this argument as follows. Assume as in [20] that packet losses to flow i can be modeled as a Poisson (counting) process $D_i(t)$ with rate $E(D_i(t)) = \lambda_i(t)$. Assuming that sources are homogeneous and setting $R_i(\bar{q}(t) = R(t)$, we can express the behavior of Binomial-TCP's congestion window control as

$$dW(t) = \frac{aW^n(t)}{R(t)}dt - bW^m(t)dD(t),$$

where $dD(t) = 1$ corresponds to the detection of a packet mark/drop. The parameters $(a, n; b, m)$ are described in Section III-A. TCP, for example, has parameters $(1, 0; .5, 1)$.

[13] Or, a marked packet is more likely to belong to a flow with a larger-than-average-sized window. This is a consequence of the Inspection paradox (e.g. see p. 65 of [31]).

	a	n	b	m	$\bar{W}/\bar{W}max$ (theory)	$\bar{W}/\bar{W}max$ (sim)
TCP	1	0	0.5	1	0.750	0.733
AIMD	0.25	0	0.75	1	0.625	0.617
Binomial	1	1/3	0.5	1	0.740	0.713

TABLE II

$\bar{W}$ AS A FUNCTION OF $\bar{W}max$ FOR VARIOUS SCHEMES

If we take expections on both sides of the above equation, and make the approximation as in [20] that $E(\bar{W}^n) \approx (E\bar{W})^n$, we obtain

$$d\bar{W}(t) = \frac{a\bar{W}^n(t)}{R(t)}dt - bE\left(W^m(t)dD(t)\right). \qquad (12)$$

Let $\bar{W}max$ be the expected window size, taking the expectation over those instants just before the window size is cut in response to a mark/drop. This gives

$$\frac{d\bar{W}(t)}{dt} = \frac{a\bar{W}^n(t)}{R(t)} - b\bar{W}^m_{max}(t)\lambda(t).$$

We still need to calculate the relationship between $\bar{W}$ and $\bar{W}max$. First note that

$$\bar{W} = \frac{\int_{\bar{W}_{max}-b\bar{W}^m_{max}}^{\bar{W}_{max}} W(t)dt}{T},$$

where T is the average time interval between two successive drops. Note also that the following are true:

$$\int_{\bar{W}_{max}-b\bar{W}^m_{max}}^{\bar{W}_{max}} W(t)dt = \int_{\bar{W}_{max}-b\bar{W}^m_{max}}^{\bar{W}_{max}} \frac{W}{\frac{dW}{dt}}dW,$$

$$T = \int_{\bar{W}_{max}-b\bar{W}^m_{max}}^{\bar{W}_{max}} \frac{1}{\frac{dW}{dt}}dW.$$

Over the range of the integral,

$$\frac{dW}{dt} = \frac{aW^n(t)}{R(t)}.$$

Assuming $R(t)$ stays constant, we derive that

$$\bar{W} = \frac{\int_{\bar{W}_{max}-b\bar{W}^m_{max}}^{\bar{W}_{max}} W^{1-n}dW}{\int_{\bar{W}_{max}-b\bar{W}^m_{max}}^{\bar{W}_{max}} W^{-n}dW}. \qquad (13)$$

We have calculated the relationship between $\bar{W}$ and $\bar{W}max$ for several TCP schemes, and give the results in Table II. The table also lists the empirical evaluation of this ratio for a trace generated using ns. For normal TCP, (13) gives $\bar{W} = \frac{3}{4}\bar{W}_{max}$ which agrees with our earlier (less rigorous) argument.

To test this modified fluid model, we ran an ns simulation of the basic setup of Section III-A, and compared these results to those found by numerically solving the fluid equations. Figure 33 shows that our modified fluid equation is more accurate than that proposed in [20]. We have also compared our modified equation to that of [20] for the simulation scenario described in Figure 3(a) of [20], and found that while both models closely match ns simulations, ours is a slightly tighter fit.

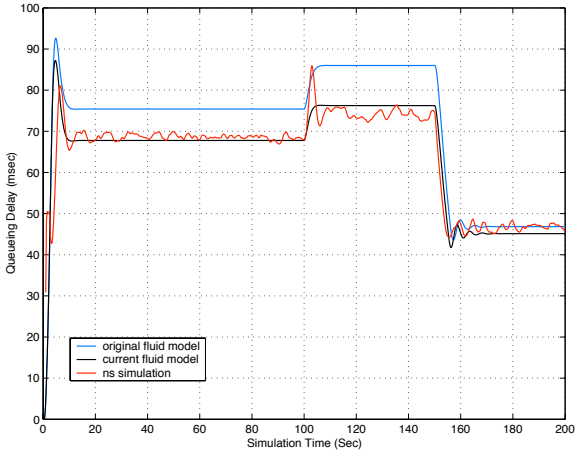

Fig. 33. Fluid model comparisons at Q1 for Normal TCP