



TCP slow start with fair share of bandwidth

InKwan Yu^{a,*}, Richard Newman^b

^a The Institute of Computer Information & Communication, Korea University, Seoul, Republic of Korea

^b Dept. of CISE, University of Florida, Gainesville, FL, USA

ARTICLE INFO

Article history:

Received 27 May 2010

Received in revised form 9 August 2011

Accepted 15 August 2011

Available online 26 August 2011

Keywords:

Network

TCP

Slow start

ABSTRACT

The initial start-up performance of TCP largely depends on two parameters – *ssthresh* and *cwnd*. When these values are not accurate, TCP cannot utilize the bandwidth fully or may generate multiple packet drops. Unfortunately, estimating these parameters are not easy, since little network state information is available for the TCP connection initially.

From earlier research, a TCP parameter of a previous connection with the same destination was suggested to be used for a new TCP connection. However, the effectiveness of this method is limited, since the cached parameter of single destination is used. As another attempt, a network monitor was adopted to identify the connections sharing the same subnet, and an averaged parameter of those connections was used for a new TCP connection. In this approach, the overhead of the network monitor may be high.

In this paper, fairness of TCP connections sharing the same bottleneck links is considered in obtaining *ssthresh* and *cwnd* of a new TCP connection without a network monitor. For evaluation, these parameters are used in simulation with four different slow start strategies, namely, LISS, ISS, MISS, and JS, depending on which parameters are used and whether packet pacing is used. The simulation results show that our estimation method works well for homogeneous and moderately heterogeneous environments.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

In TCP [1], slow start uses the slow start threshold (*ssthresh*) as a parameter for the upper bound of rapid congestion window growth. The value of *ssthresh* significantly impacts the performance of TCP. When it is too small, the slow start phase terminates too early, losing the opportunity to utilize available bandwidth. When it is too large, slow start induces multiple packet losses before it reaches *ssthresh*. It is obvious that a good *ssthresh* value early in the slow start phase can significantly improve TCP performance.

First, we introduce a new method to obtain *ssthresh*. This method, which we call *Less Informed Slow Start* (LISS), determines *ssthresh* using the congestion window (*cwnd*) of the oldest connection among connections sharing com-

mon bottlenecks. Assuming fairness of TCP [2], LISS is relatively accurate, although its application is rather limited since it requires steady and active TCP connections sharing bottlenecks. In other words, our scheme assumes that a new TCP connection will use the same bandwidth as other TCP connections sharing bottleneck links and assigns the *ssthresh* value according to this bandwidth. It should be noted that LISS improves TCP Reno [3] without modifying the TCP algorithm itself, as it only changes a TCP parameter. If it is allowable to modify the TCP algorithm, further improvement over LISS is attainable. When estimating *ssthresh*, Exponentially Weighted Moving Average (EWMA) of *cwnd* of the oldest connection among connections sharing common bottlenecks, can be used instead of *cwnd*, as in the ns-2 [4] simulator, to smooth out the typical *cwnd* growth pattern that follows the saw-tooth shaped cycles. We call this slow start scheme *Informed Slow Start* (ISS). Related to LISS and ISS, Hoe [5] uses the original TCP slow start algorithm with *ssthresh* estimated with a packet pair.



* Corresponding author.

E-mail address: inkwan@gmail.com (I. Yu).

Both LISS and ISS use the standard initial value of *cwnd*, which is one. In addition to the value for *ssthresh* obtained by LISS, it would be beneficial if we can estimate an initial *cwnd* value that would not induce packet drops. If a new TCP connection can start with the estimated *cwnd* value, slow start can increase *cwnd* rapidly until *ssthresh* is reached. We introduce a novel method in Section 4 to estimate such a value for *cwnd* based on the states of the existing TCP connections. As this scheme relies on estimation of both *cwnd* and *ssthresh*, it will be called *More Informed Slow Start* (MISS).

As yet another approach, the initial *ssthresh* obtained in LISS is also used as the initial *cwnd*. We call this scheme *Jump Start* (JS), as it does not involve exponential growth of *cwnd*. JS is similar to the work of Zhang et al. [6], as their algorithm also bursts initial *cwnd* packets with packet pacing. However, in their algorithm, the initial *cwnd* value of a new connection is taken from a moving average of *cwnd* values of other connections sharing bottleneck links estimated by a network monitor. In both MISS and JS, in order to reduce the impact of overshooting packets, packet pacing as in Padmanabhan and Katz [7] is employed.

To summarize, for a new connection, LISS is the same as TCP Reno's slow start algorithm except that *ssthresh* is taken from *cwnd* of the TCP connection with the longest duration sharing the same bottlenecks. ISS differs from LISS in that *ssthresh* is obtained from an EWMA of *cwnd* from the TCP connection with the longest duration. MISS uses *ssthresh* as in LISS and, in addition, obtains the initial *cwnd* of new connection estimated from Eq. (5). JS sets the initial *cwnd* value from *ssthresh* of LISS. Both MISS and JS adopt packet pacing for initial *cwnd* packets.

All these approaches apply to a server that handles multiple clients. By using the IP address prefixes to relate connections, the server has the opportunity to obtain the information about connections that may share common bottlenecks.

The performance of our schemes is influenced by fairness of TCP. Fairness is achieved if each connection with common bottlenecks has the same bandwidth. It seems that fairness is harder to observe with greater cross traffic and larger heterogeneous networks. In general, TCP performance degrades with a long delay paths. Consequently our scheme works better with a small and homogeneous network.

In this study, without losing the general principles of TCP, we assume that *ssthresh* represents the number of packets, and slow start terminates if *cwnd* reaches *ssthresh*, whereas, with the standard TCP, *ssthresh* is represented in bytes.

2. Related work

In TCP Reno, slow start increases *cwnd* by one for every acknowledgement (ACK) the TCP sender receives from the TCP receiver, until *ssthresh* is reached. The default *ssthresh* value is fixed at 65,535 bytes. This fixed *ssthresh* value is chosen in the absence of measurement information at the initial stage of a TCP connection. Evidently, it is desirable to estimate initial bandwidth to obtain a more accurate

ssthresh value in order to improve slow start performance. To estimate the end-to-end bandwidth, there have been a number of research efforts [8,9,5,10–13].

A weakness of the current TCP slow start mechanism becomes apparent, when there is a large delay bandwidth product ($\text{delay} \times \text{bandwidth}$) path. In a network path with a large round trip time (RTT) value and high bandwidth, slow start is not fast enough. For example, it takes a long time to increase *cwnd* for a typical satellite network [14]. In TCP Reno, self-clocking of packets is used while *cwnd* is limited by *ssthresh* [15]. Due to self-clocking, the *cwnd* size does not grow fast enough if RTT is large, even though the congestion window growth rate is exponential.

If the initial *ssthresh* value is small in a large delay bandwidth product path, the slow start phase terminates too early, and then the *cwnd* increases slowly under the Additive Increase and Multiplicative Decrease (AIMD) phase of TCP Reno. To alleviate the small initial window size problem, some modifications have been suggested [14,16]. Nevertheless, the fixed initial window size is still a problem [17]. If we have a good estimate of available bandwidth, it is reasonable to increase *cwnd* to the available bandwidth quicker than TCP Reno.



Visweswariah and Heidemann [18] investigate the reuse of previous packet transmission rate or the congestion window size of an idle TCP connection. In their work, simulation results show that pacing packets using the rate before the idle period performs better than the standard slow start algorithm or bursting packets, when the connections have been idle for less than a minute. Even though it is not shown in the paper, if the idle time is more than a few minutes, it may not be effective as the previous window size would be stale for the current network state. TCP Fast Start introduced in Padmanabhan and Katz [7] uses a previous connection's cached connection information for a new connection with the same destination address. In TCP Fast Start, multiple packet drops may occur, if the cache becomes stale and inaccurate. In contrast to Visweswariah and Heidemann [18] and TCP Fast Start [7], where only one connection is considered in estimation, our approach uses other connections sharing bottleneck links with a new connection to determine the *cwnd* and *ssthresh* values for the new connection.

Zhang et al. [6] is similar to our approach in that they use other connection information with the same destination subnet. However, their algorithm requires a network monitor called the *performance gateway* to aggregate information and to estimate the *cwnd* value for a new connection, while we use TCP fairness to estimate *cwnd* without a network monitor.

To accelerate *cwnd* growth, the slow start transmission rate can change based on the amount of available bandwidth [17]. When there is more available bandwidth, more packets are sent. Several ways of increasing *cwnd* are suggested [14,16,17].

Slow start may occur in three different stages of a TCP connection – when the connection is initially established, when the connection is idle for a while, and when there is a timeout [16]. These three cases can be handled differently since the connection information available for each

case is different. There have been efforts to solve each case in the literature [19,8,7,18,17]. Among these three cases, the latter two have an advantage in that they can easily acquire the states of the current connection such as the congestion window size, RTT, and its variance.

For the initial slow start, it's hard to obtain an accurate estimate of available bandwidth. Hoe [5] uses a packet pair method to estimate the initial *ssthresh* value. However, the inter-packet gap of the first single packet pair is not accurate enough due to multiple hops in the path and measurement error. Furthermore, the first inter-packet gap causes overestimation of initial congestion window size [20,11,21,22]. Multiple packet pairs are used to estimate the *ssthresh* value in Hu et al. [23] and Aron and Druschel [8]. In Hu and Steenkiste [23], a variant of the slow start algorithm detects the change between inter-packet gaps when the connection reaches the peak available bandwidth. The authors also use packet pacing to reduce the impact of slow start on routers.

In another approach, TCP Westwood [17] uses an improved bandwidth estimation method of TCP Vegas [24]. This approach dynamically adjusts the slow start packet transmission rate. When there is more available bandwidth, the sending rate increases more rapidly and conversely, when there is less bandwidth available, the sending rate decreases. Although not directly related to slow start, there have been a number of studies calculating available bandwidth [20,25,21,22,26,13]. Most of these schemes use trains of packets to measure the bandwidth more accurately.

As for the initial *cwnd*, Allman [14] shows that it is beneficial to use a large initial *cwnd* for certain cases. Zhang et al. [6] introduce a scheme to speed up the transfer of small files using TCP. They take a moving average of *cwnd* values of other connections sharing bottleneck links and the file size to be transferred, in calculating the initial *cwnd* value. This scheme is similar to JS in that it also bursts *cwnd* number of packets with packet pacing initially, however, it differs in the *cwnd* estimation method.

The idea of using information of other connections sharing bottleneck links has been around for nearly two decades. Savage et al. [27] show strong evidence of locality among network connections. This locality facilitates “informed congestion control” of other connections with the same locality. For the purpose of collecting information of connections with shared bottlenecks, a passive monitor can be adopted. Then, this information can help other connections in making congestion control decisions.

Balakrishnan et al. [28] introduce a scheme called *Congestion Manager* (CM) to aggregate connection information in the OS kernel residing in between TCP and IP stacks. In addition, CM behaves as mediator of TCP and UDP flows to provide better congestion control performance using the information from multiple flows sharing the same network characteristics.

In this paper, by way of simulation, the initial slow start performance is measured during 1 s through 10 s. Selvidge et al. [29] mention that users tend to lose interest if they have to wait more than 10 s to download a web document. Wang [30] also finds that the actual delay affects user experience significantly.

3. Estimating congestion window size for slow start threshold

When a TCP connection is already established from a source to a destination, it is reasonable to assume that the bandwidth of the initiating connection from the same source to the same destination should not exceed the stable connection bandwidth, if TCP is fair in the sense that the protocol allocates the same bandwidth for each connection. Also, it can be assumed that the new connection bandwidth would not be lower than the half of the stable connection bandwidth. Thus, the new connection bandwidth would follow the following inequality:

$$\frac{b_o}{2} \leq b_n \leq b_o, \quad (1)$$

where b_o is the stable connection bandwidth, and b_n is the new connection bandwidth sharing the same source and destination. The latter equality of Eq. (1) occurs when the capacity of bottleneck is larger than the additional bandwidth required by the new connection. To generalize this, define C as the bottleneck link capacity, B_i as a connection currently passing through the bottleneck link, and let b_i be the bandwidth of B_i , where $i \in \{1, 2, 3, \dots\}$. Let B_o be the stable connection with long duration and B_n be the new connection. If $C \geq \sum_i b_i + b_n$ with $\{B_o\} \subset \cup \{B_i\}$ and $\{B_n\} \not\subset \cup \{B_i\}$, then with fair TCP, $b_n = b_o$. The first equality of Eq. (1) can happen when $C = b_o$, and $\cup \{B_i\} = \{B_o\}$. In this case, with addition of B_n , we have $C = b_o/2 + b_n$ and $b_n = b_o/2$. More generally, if $C = \sum_i b_i$, and a new connection is added, its bandwidth will be reciprocal to the number of connections in the bottleneck link. Hence, if the number of stable connections is n , then the minimum available bandwidth for the new connection is

$$b_n = \frac{nb_o}{n+1}.$$

The same idea can be generalized when a set of hosts share a common set of bottlenecks. As heuristics, it is possible to assume that hosts in the same subnet share a set of bottlenecks to a single source. In this case, we can further expect that connections from the same subnet to the same source behave just like connections from a single host. Furthermore, if bottlenecks exist between ISPs and not within an ISP and delay within the ISP is negligible, the connections from the same ISP will behave similarly as a single host.

If β_n is the new connection bandwidth from a LAN or an ISP and β_o is stable connection bandwidth in the same LAN or the ISP with fair TCP, and the source can determine a set of destinations passing through the shared bottleneck links, then the minimum bandwidth of new connection will be

$$\beta_n = \frac{K\beta_o}{K+1},$$

where K is the total number of connections sharing the bottleneck links.

In practice, an accurate value for the available bandwidth is not readily available. To avoid making changes to the TCP Reno implementation, we take *cwnd* of an

existing TCP connection in the AIMD phase as an estimate of available bandwidth. If there are multiple live connections sharing bottlenecks, $cwnd$ of the oldest connection is taken. Selecting the oldest connection can be easily implemented using a queue and gives a better bandwidth estimate as TCP connections take time to reach steady state. Among live connections, ones under the slow start phase should be separated from those under the AIMD phase. Connections in the slow start stage are not stable and should not be assumed to take the same bandwidth as other, steady connections.

Now, in light of the above discussion, a practical $ssthresh$ estimation scheme can be given. First, let w_o be the congestion window size of the oldest steady connection and s_n be the $ssthresh$ value of the new connection. The number of active TCP connections in the AIMD phase is represented as L and the number of active TCP connections in the slow start phase as M . Define $cwnd$ of a single TCP connection under the slow start phase as σ_i , $i \in \{1, 2, 3, \dots, M\}$ and the sum of them as $S = \sum_{i=1}^M \sigma_i$. Then, the $ssthresh$ value for a new connection is estimated by

$$s_n = \frac{S + Lw_o}{L + \frac{S}{w_o} + 1}. \quad (2)$$

In Eq. (2), $S + Lw_o$ estimates the sum of $cwnd$ s of TCP connections sharing the same bottlenecks. To get a fair congestion window size estimate for a new connection, the sum is divided by the number of TCP connections plus one. However, TCP connections under the slow start phase are not in a stable state and their $cwnd$ values are not fair compared to other steady TCP connections. If w_o is a fair $cwnd$ value, $\frac{S}{w_o}$ can estimate the number of steady connections, assuming S is used for steady connections instead of start-up connections.

The estimation of $ssthresh$ described above depends on w_o in Eq. (2). If we take $cwnd$ of the oldest connection as w_o , slow start would be LISS. Instead, if $awnd$ is taken for w_o as given in the pseudo code below, then slow start would be ISS.

$$awnd \leftarrow (1 - \alpha) \cdot awnd + \alpha \cdot cwnd.$$

JS uses $ssthresh$ obtained in LISS for both initial $cwnd$ and $ssthresh$ values of JS. This way, JS avoids exponential growth of $cwnd$ and starts to transmit $cwnd$ number of packets in a burst. To reduce the impact from the burst, JS adopts packet pacing.

4. Estimating initial congestion window size

Slow start performance is influenced by the $ssthresh$ value to increase $cwnd$ rapidly. Also the performance depends on the initial $cwnd$ value. To motivate our approach, assume that there is only one TCP connection between the TCP sender and the TCP receiver with a constant FTP feed. As there is no competing traffic, the $cwnd$ size of this TCP connection in congestion avoidance phase will linearly increase until no more bandwidth is available, whereupon a packet drop occurs and $cwnd$ shrinks to half of its size. When the linear growth rate of $cwnd$ is $1/R$, where R is the RTT of the connection [31], we'd like to know a good

initial $cwnd$ value for a new connection that shares the same source and destination.

If we know the packet drop interval of the existing connection and the last time that a packet drop occurred, it is possible to estimate a good initial $cwnd$ value. Let δ be the packet drop interval and the time of last packet drop be t_l . If the current time when a new connection starts is t_c , then $t_c - t_l$ represents the amount of time since the last packet drop until now and $\delta - (t_c - t_l)$ becomes the amount of time until the next packet drop. As the existing TCP connection increases $cwnd$ during this period with the rate of $1/R$,

$$\frac{\delta - (t_c - t_l)}{R},$$

can be a good initial $cwnd$ value for the new connection.

This idea can be generalized for TCP connections with shared bottleneck links. Let

$$T = \{t_i | i = 1, 2, 3, \dots, l \text{ and } t_j < t_k, j < k\},$$

be the ordered set of packet drop times of all connections sharing the same bottleneck links. Also let

$$U = \{u_i | i = 1, 2, 3, \dots, m \text{ and } u_j < u_k, j < k\},$$

be the set of packet drop time of the oldest connection among connections sharing the same bottleneck links with $U \subset T$. For example, in Fig. 1, it is assumed that two TCP flows F_1 and F_2 share the same bottleneck links. As F_1 is the oldest connection among F_1 and F_2 , u_i refer to packet drop times of F_1 . Also, packet drop times of both connections are labeled with t_i , where u_i coincide with some of t_i .

If we define the packet drop interval of connections with shared bottlenecks as $t_{i+1} - t_i$, where $t_{i+1}, t_i \in T$, then EWMA of shared connections' packet drop interval would be

$$\tau_i = (1 - \lambda)\tau_{i-1} + \lambda(t_i - t_{i-1}), \quad (3)$$

where $2 \leq i \leq l$, $0 \leq \lambda \leq 1$ and $\tau_1 = t_2 - t_1$. Likewise, for $u_{i+1}, u_i \in U$, when the packet drop interval of the oldest connection among connections with shared bottlenecks is $u_{i+1} - u_i$, EWMA of the oldest connection packet drop interval would be

$$\gamma_i = (1 - \theta)\gamma_{i-1} + \theta(u_i - u_{i-1}), \quad (4)$$

where $2 \leq i \leq m$, $0 \leq \theta \leq 1$ and $\gamma_1 = u_2 - u_1$.

If we take γ_m/τ_l as the number of connections sharing bottleneck links with the same $cwnd$ growth rate, and each connection $cwnd$ growth rate is $1/R$, then

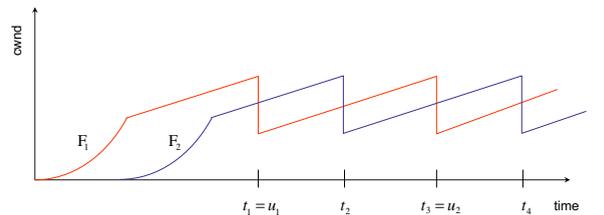


Fig. 1. $cwnd$ graphs of two TCP flows F_1 and F_2 sharing the common bottlenecks.

$$\frac{\gamma_m(\tau_l - (t_c - t_l))}{\tau_l R}, \quad (5)$$

can be a candidate *cwnd* size for a new TCP connection sharing the same bottleneck links.

This is reasonable as the number of packets estimated in Eq. (5) is the amount of packets we may add on the path with shared bottlenecks without causing a packet drop. In MISS, we take *cwnd* for a new connection as estimated above and *ssthresh* as obtained in LISS, and to be conservative when *cwnd* > *ssthresh*, we set the *ssthresh* value to *cwnd*. Furthermore, to reduce the impact of the initial start-up burst of packets, the first *cwnd* packets are paced with the packet transmission rate *cwnd*/*R*.

5. Simulation

To compare these approaches to improving early TCP performance, let *T* be the total number of connections established between the server and clients during the simulation, and *c_i* be an instance of such connection with *i* ∈ {1, 2, 3, ..., *T*}. The number of packets acknowledged of the connection *c_i* by time *t* is represented as *c_i^t*. Here, we assume that the unit of time is the second. Then, $\sum_{i=1}^T c_i^t$ is the number of packets acknowledged by the server during the first *n* seconds of each connection, which we call the *initial n second throughput*. In comparison, the total number of packets acknowledged by the server during the whole simulation is called the *total throughput*. Occasionally, these two throughput criteria are collectively called the *server throughput*.

Usually, networks with large delay bandwidth product paths benefit from LISS and ISS in terms of the server throughput. Depending on the network topology and traffic characteristics, the degree of performance gain of LISS and ISS against TCP Reno may differ significantly. With a homogeneous network, LISS and ISS show much higher throughput than TCP Reno since fairness of TCP is well preserved in such an environment. However, LISS and ISS achieve only slightly better throughput compared to TCP Reno in a large and heterogeneous network.

In contrast, MISS and JS show significant performance improvement over LISS and ISS in terms of the *n* second throughput for connections with long duration. However, as bandwidth increases, MISS and JS perform slightly worse in terms of the total throughput even though the initial *n* second throughput shows impressive improvement,

when the router queue size is limited. Unfortunately, for connections with short duration, the MISS performance drops significantly as a good *cwnd* value cannot be obtained from connections with short duration.

To evaluate each model, we simulate each scheme using ns-2 [4] version 2.28 on Linux. As there is no tear down mechanism for one-way TCP connection in ns-2, *reset* OTCL command is added to the TCP agent class. Terminated TCP connections are reset and reused later as a new connection needs to be established between the same sender and the receiver. In ns-2, the default value of *ssthresh* is initialized to the default advertised window size, which is undesirable for our purpose. Hence, we decouple these two parameters by initializing the default advertised window size to 100 and the default slow start threshold value to 10. Also, by default, *cwnd* is set to one. These default initial *cwnd* and *ssthresh* values are used for TCP Reno and the default initial *cwnd* value is used for ISS and LISS. For all simulation scenarios, the TCP receiver is not enabled with delayed acknowledgement. A new OTCL command in ns-2 is added to set the *ssthresh* value for the TCP sender to use the obtained *ssthresh* value. When there is no other active TCP connections sharing bottlenecks, the default *ssthresh* value is used. *awnd* is predefined in ns-2 where the EWMA weight parameter α of *awnd* is set to 0.002 by default. EWMA weight parameters of Eqs. (3) and (4) are set to 2^{-3} for ease of implementation.

5.1. Homogeneous network

First, a homogeneous network is considered, where the effect of TCP fairness is noticeable and the obtained *ssthresh* is relatively accurate as clients share the same network characteristics. The topology used for the simulation is shown in Fig. 2. There is a single server and five clients each labeled with “s” and with “c”, respectively. Routers between them are labeled as “r1”, “r2”, and “r3”. To generate cross traffic between routers r1 and r2, five pairs of source and destination are used. Nodes labeled with “xs” are cross traffic sources and nodes with “xc” are corresponding destinations in the figure. “Delay” labeled on the links between r1 and r2, as well as r2 and r3 is a parameter used in the simulation to assign different link delays. Similarly, “Bandwidth” is a parameter for link bandwidth.

For cross traffic, FTP traffic is generated between five (xs, xc) pairs. To alleviate the effect of simultaneous connection establishments, FTP connections arrive with the

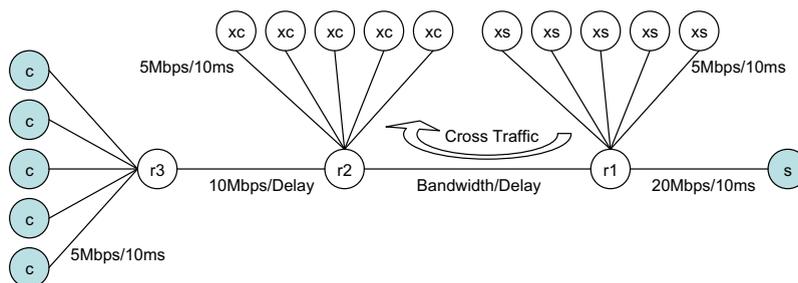


Fig. 2. Homogeneous network topology.

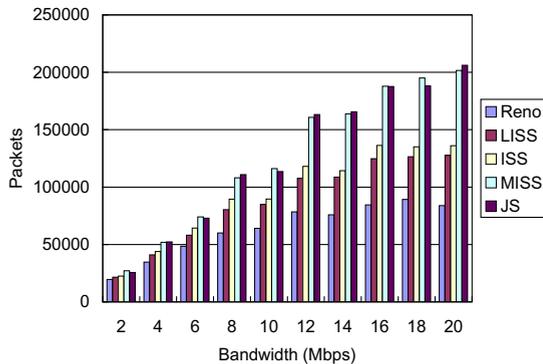
exponential distribution average of 1 s. Queueing discipline for routers is drop-tail with 50 packets as the maximum queue size. Other router parameters are default values in ns-2. FTP connections from the server to clients arrive with the exponential distribution average of 5 s while each client may have multiple FTP connections with the server. Unless mentioned, Delay is set to 50 ms and Bandwidth is set to 20 Mbps in Fig. 2. Duration of connections between the server and clients follows the Pareto distribution with a mean of 100 s and shape parameter of 1.35. The simulation runs for 3000 s.

Based on the simulation setup given above, we first evaluate how each model behaves with varying Bandwidth values. In Fig. 3, TCP Reno, LISS, ISS, MISS and JS are compared in terms of the server throughput. Fig. 3b shows the total number of packets acknowledged by the server, i.e., the total throughput with varying Bandwidth, whereas Fig. 3a shows the initial 5 s throughput. Clearly, there is significant difference in the initial 5 s throughput as Bandwidth increases. Even though there is not so much difference of the total throughput, it is visible that as Bandwidth reaches 18 Mbps, MISS and JS throughput becomes slightly worse. This can be attributable to the initial burst transmission of packets in MISS and JS with higher

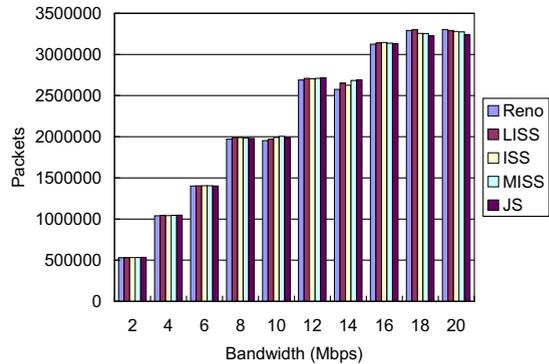
capacity links, even when they try to reduce the impact by packet pacing.

In Fig. 3a, LISS, ISS, MISS and JS all show better performance than TCP Reno in terms of the initial 5 s throughput with comparable performance of the total server throughput. With a link speed of 16 Mbps, TCP Reno successfully delivers 84,545 packets to clients, while 124,719, 136,307, 187,868, and 187,678 packets are delivered by LISS, ISS, MISS, and JS, respectively. This improvement amounts to more than 100% for MISS and JS against TCP Reno.

Fig. 4 shows the initial 5 s throughput and the total throughput with varying Delay. It is well known that TCP throughput decreases as RTT increases and Fig. 4b shows this tendency clearly. Notice that LISS, ISS, MISS, and JS outperform TCP Reno in terms of the total throughput as Delay increases even though the total throughput of JS approaches that of TCP Reno as Delay increases. With large Delay, the initial *cwnd* and *ssthresh* values have greater influence on the total throughput of TCP. The initial 5 s throughput shows better performance when Delay is relatively small for LISS and ISS compared to TCP Reno, while MISS and JS show much better performance achieving more than 30 times the initial 5 s throughput of TCP Reno

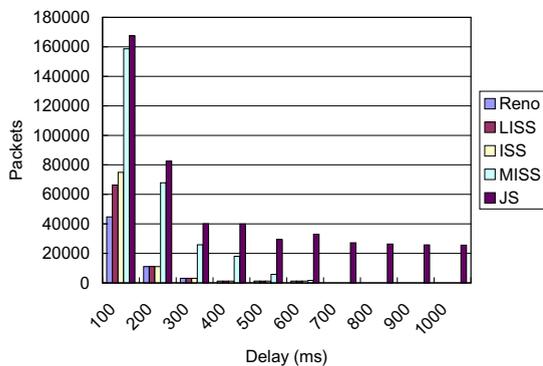


(a) Initial 5 second throughput.

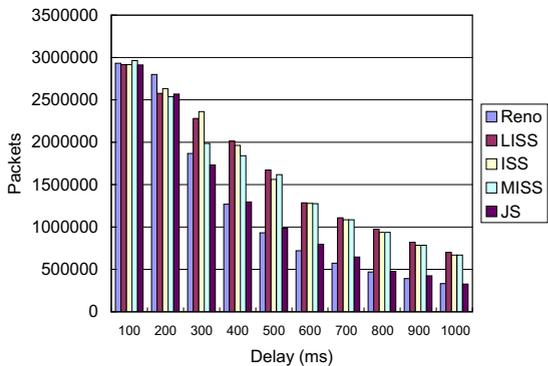


(b) Total throughput.

Fig. 3. Server throughput with varying Bandwidth in homogeneous topology.



(a) Initial 5 second throughput.



(b) Total throughput.

Fig. 4. Server throughput with varying Delay in homogeneous topology.

in some cases. This implies that the initial *cwnd* value impacts the initial throughput of TCP significantly with long delay paths.

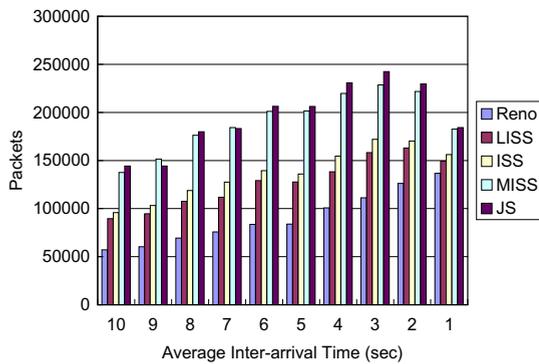
Good performance of LISS, ISS, and MISS compared to TCP Reno at small delay is probably due to self-clocking of TCP Reno. Self-clocking limits the growth of *cwnd* during the slow start phase. For example, during 5 s, there can be 10 self-clocking cycles for a connection with 500 ms RTT while there can be 100 self-clocking cycles for a connection with 50 ms RTT. This implies that the connection with 50 ms RTT has 10 times more opportunities to increase its congestion window size while in the slow start phase.

Fig. 5 shows the initial 5 s throughput and the total throughput with varying average FTP connection inter-arrival time between the server and clients. The inter-arrival time follows the exponential distribution. The initial 5 s throughput is apparently superior with LISS and ISS to TCP Reno, while that of MISS and JS is even more superior. The total throughput shows that LISS and ISS perform slightly worse than TCP Reno, while MISS and JS perform even worse. When the link capacity is higher, large *cwnd* and *ssthresh* values for new TCP connections can cause packet drops in the router with a small queue size, decreasing the total throughput. As evidence, in Fig. 6,

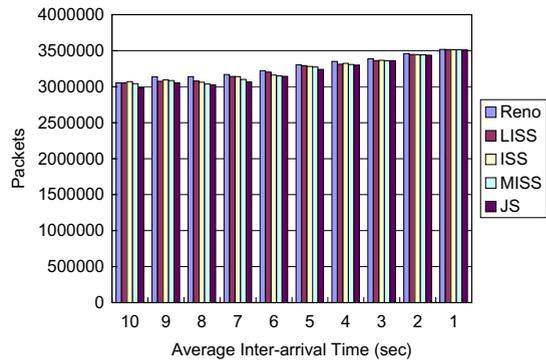
the Bandwidth value is set to 10 Mbps instead of the default value of 20 Mbps. Here, the total bandwidth of LISS, ISS, MISS and JS is comparable against TCP Reno, while the 5 s throughput shows significant improvement.

Fig. 7 shows the initial *n* second throughput where *n* ranges from 1 to 10. It is observable that the throughput increases much faster for LISS and ISS achieving 50% higher throughput than TCP Reno at 3 s, whereas MISS and JS start with more than 10 times of throughput at 1 s compared to TCP Reno. This is attributable to the fact that these two algorithms do not waste time reaching steady congestion window size.

In our simulation, the connection duration between the server and a client follows the Pareto distribution. The smaller the shape parameter of the Pareto distribution, the heavier tail the Pareto distribution has. For FTP connections with long duration, more accurate *cwnd* and *ssthresh* values can be obtained as long connections tend to be in steady state. In Fig. 8, LISS, ISS, MISS and JS achieve higher initial 5 s throughput than TCP Reno as the Pareto shape parameter gets smaller. MISS and JS show slightly worse total throughput than TCP Reno due to their aggressive *cwnd* increment policy. The shape parameter values, 1.05 and 1.95 are rather extreme, and considering that file

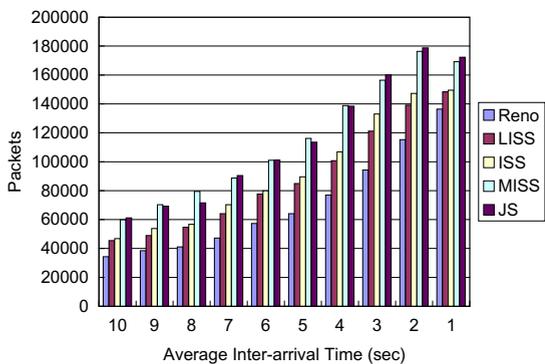


(a) Initial 5 second throughput.

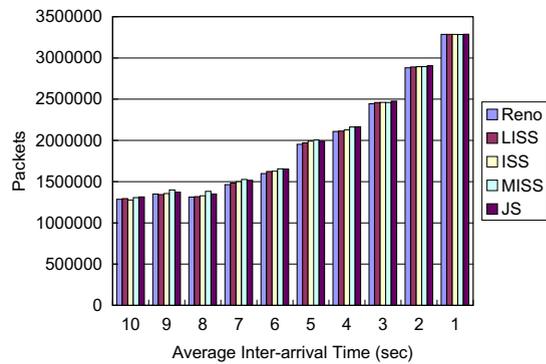


(b) Total throughput.

Fig. 5. Server throughput with varying connection inter-arrival time in homogeneous topology (20 Mbps link).



(a) Initial 5 second throughput.



(b) Total throughput.

Fig. 6. Server throughput with varying connection inter-arrival time in homogeneous topology (10 Mbps link).

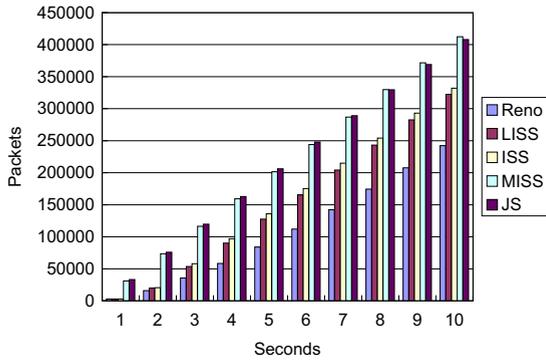


Fig. 7. Server n second throughput in homogeneous topology.

transfers would take a heavy-tailed distribution [32–34], the default parameter value of 1.35 is justified.

To summarize, it is shown that LISS, ISS, MISS, and JS outperform TCP Reno in a small homogeneous network. Especially when the bandwidth is higher and the link delay is small, these four models show significant improvement over TCP Reno for the initial n throughput. Good performance of four models is obtained, since fairness of TCP is well preserved in the homogeneous environment. However, when the router is not equipped with enough queuing buffer, packet drops can occur, especially with JS, resulting in the slightly reduced total throughput. For example, in Fig. 5b, the total throughput achieved by JS for all average inter-arrival times against that of TCP Reno is 98.1%.

5.2. Large network

For simulation of larger networks, the hierarchical addressing feature of ns-2 is used. As the hierarchical addressing is prefix-based, it is possible to identify nodes with common address prefixes similar to subnets in the Internet. In addition to hierarchical addressing, a transit-stub based random topology generator is used. For this purpose, Georgia Tech Internetwork Topology Models (GT-ITM) [35] is chosen. The ns-2 package provides a few

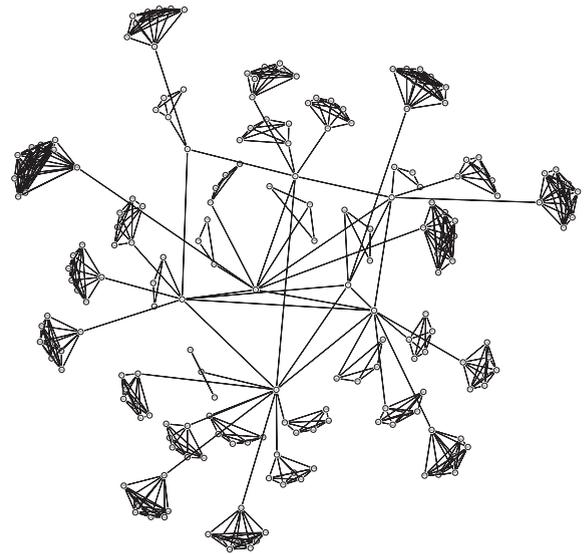


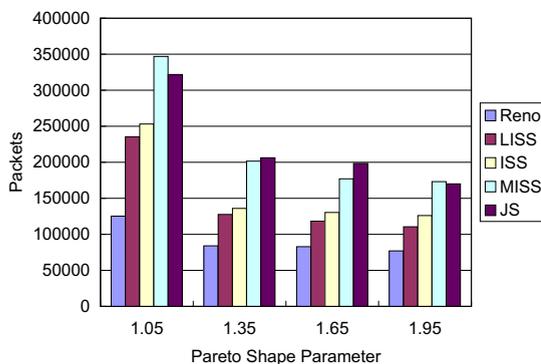
Fig. 9. Random topology with 200 nodes.

sample configurations of GT-ITM and we slightly modify two of the sample configurations, one for 200 nodes and the other for 600 nodes. The modification on the configuration increases the sub-domain connectivity to form a topology where the nodes in a sub-domain share the common bottleneck links. Network topologies generated by these configurations are shown in Figs. 9 and 10.

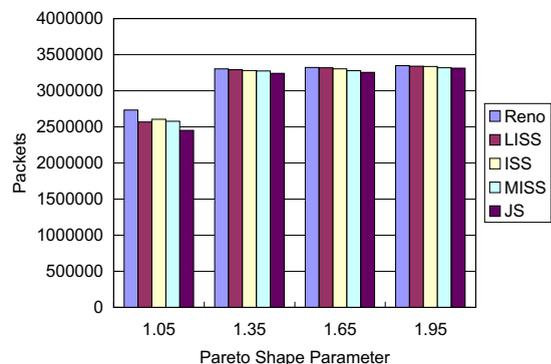
The GT-ITM topology format is converted into the ns-2 code format for hierarchical addressing, using a tool provided by the ns-2 webpage [4]. The converted ns-2 code assigns a random delay to each link while bandwidth is configurable only as a single value for all links. For simulation, we assign a server near core routers.

5.2.1. Network topology with 200 nodes

With the 200-node topology in Fig. 9, the bandwidth for all links is 10 Mbps by default and the lifetime of FTP connections between the server and clients follow the Pareto distribution with an average of 100 s and shape parameter



(a) Initial 5 second throughput.



(b) Total throughput.

Fig. 8. Server throughput with varying Pareto shape parameter in homogeneous topology.

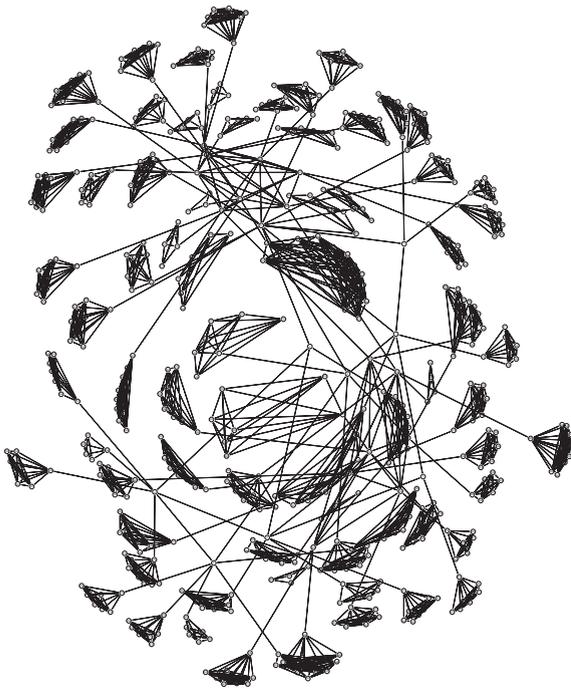
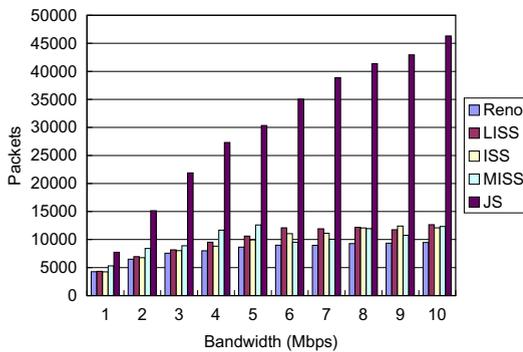


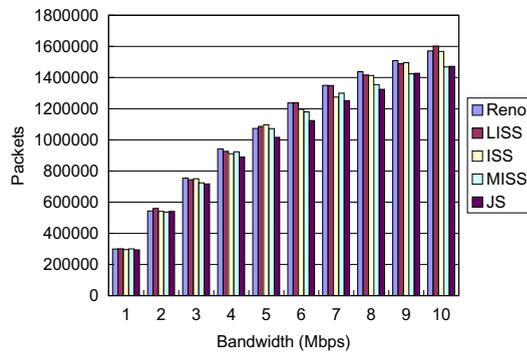
Fig. 10. Random topology with 600 nodes.

of 1.35. There are 25 random pairs of nodes generating FTP cross traffic. Each node of a pair belongs to a different sub-domain. Cross traffic connections arrive with an exponential distribution average of 1 s. Clients are randomly selected to establish connections with the server. The inter-arrival time of connections between the server and clients is distributed with an exponential distribution average of 1 s.

Fig. 11 shows the initial 5 s throughput and the total throughput with varying link bandwidth values. As the bandwidth increases, the initial 5 s throughput values of LISS, ISS and MISS increase gradually as shown in Fig. 11a. However, the total throughput of MISS for all Bandwidth values is 95.9% of that of TCP Reno. JS shows significantly better performance of the 5 s throughput than



(a) Initial 5 second throughput.



(b) Total throughput.

Fig. 11. Server throughput with varying bandwidth in 200-node topology.

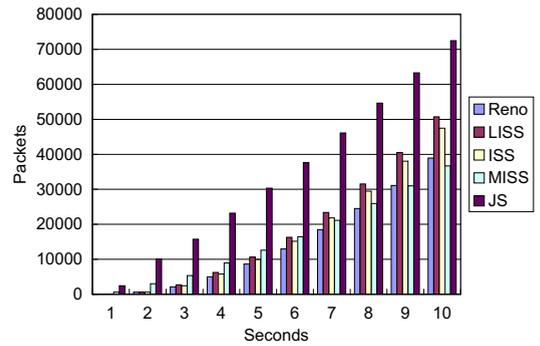


Fig. 12. Server n second throughput in 200-node topology.

TCP Reno, while its total throughput for all Bandwidth values is 93.8% of that of TCP Reno.



One noticeable tendency observed in Fig. 11b is that the total throughput of MISS and JS gets worse against TCP Reno as bandwidth increases. One reason for this may be that the initial overshooting packets of TCP connections negatively impacts the total throughput causing packet drops. Also, with the 200-node topology, the average delay is larger than the homogenous topology, which increases the initial burst of TCP causing more packet drops at the routers even with packet pacing. In addition, with heterogeneous networks, it is not easy to obtain good estimates of *cwnd* and *ssthresh*, as TCP fairness is not well maintained.

In Fig. 11a, the initial 5 s throughput is greater for LISS, ISS, MISS, and JS than for TCP Reno. As large initial 5 s throughput implies reduced initial download time for multimedia objects or web documents, users will have better experience with these four models than TCP Reno.

In Fig. 12, the initial n second throughput is shown as n ranges from 1 to 10 for the topology with 200 nodes in Fig. 9. Parameters of simulation are the same as in Fig. 11. MISS and JS achieve higher initial n second throughput for small n , as they start with large *cwnd* values. In contrast, the initial n second throughput of LISS and ISS starts with small values. Obviously, aggressive behavior to increase *cwnd* rapidly in MISS and JS helps to achieve higher throughput initially. However, this is not

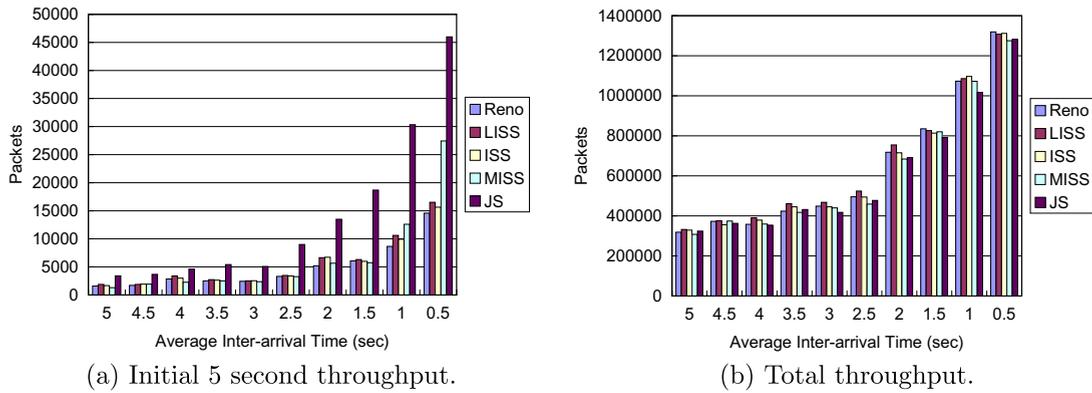


Fig. 13. Server throughput with varying exponential inter-arrival time in 200-node topology.

without cost, as the total throughput of MISS is 94.2% of TCP Reno, and the total throughput of JS is 95.7% of TCP Reno, when the total throughput for all Bandwidth values are added.

Fig. 13 shows the initial 5 s throughput and the total throughput with varying average connection inter-arrival time between the server and clients. As the average inter-arrival time of exponential distribution ranges from 5 to 0.5, LISS, ISS, MISS, and JS show increasing gain over TCP Reno in terms of the initial 5 s throughput. This is reasonable as there are more connections with common bottleneck links, so *cwnd* and *ssthresh* values can be determined more accurately. However, a large initial *cwnd* value causes overshooting of packets, inducing packet drops at the router with small buffer size. For this reason, MISS and JS show poor total throughput performance when the average inter-arrival time is small.

Since the topology with 200 nodes in Fig. 9 has larger average delay than the homogenous network in Fig. 2, aggressive growth of *cwnd* for a new connection could be detrimental to TCP start-up performance, whereas good estimates of *cwnd*, *ssthresh* and packet pacing are beneficial. Even with packet pacing, a sudden burst of packets tends to generate packet drops in the router with small queue for the topology with 200 nodes, as large average delay of the topology tends to introduce large initial *cwnd* values.

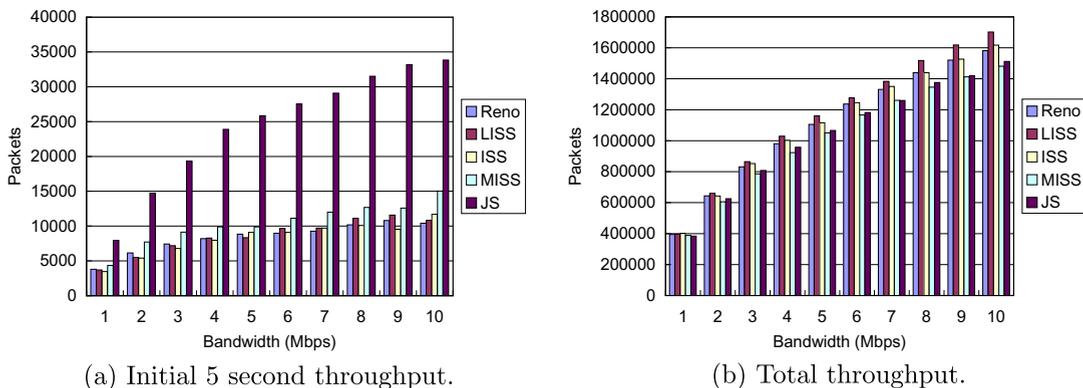


Fig. 14. Server throughput with varying bandwidth in 600-node topology.

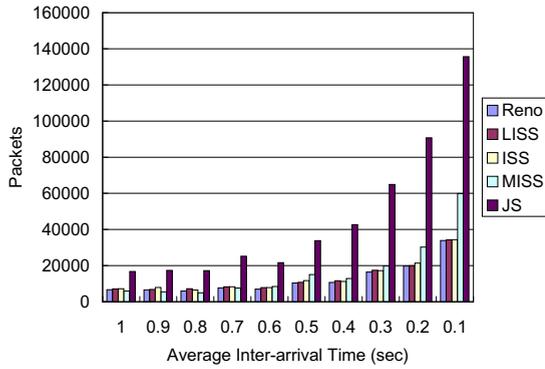
5.2.2. Network topology with 600 nodes

For simulation of topology with 600 nodes in Fig. 10, the default bandwidth for all links is 10 Mbps. The number of pairs of nodes used for cross traffic is 100. Cross traffic connection arrivals follow the exponential distribution with average of 1 s. FTP connections between the server and clients terminate with the average lifetime of 100 s. This lifetime duration follows the Pareto distribution, where the shape parameter of Pareto distribution is 1.35. The average inter-arrival time of connections between the server and clients is 0.5 s by default.

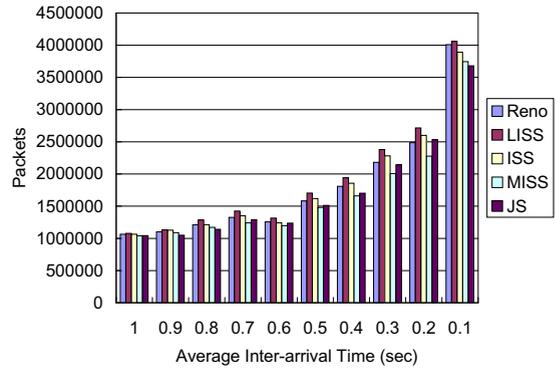
For varying link bandwidth values, Fig. 14 illustrates TCP Reno, LISS, ISS, MISS and JS in terms of the initial 5 s throughput and the total throughput. In Fig. 14a, it is not clear which one performs better in terms of the initial 5 s throughput among LISS, ISS and TCP Reno, whereas MISS and JS significantly outperform TCP Reno. However, MISS and JS achieve this by sacrificing the total throughput.

As shown earlier, estimation of *ssthresh* becomes more difficult as the network size grows, since fairness of TCP is harder to achieve in large and heterogeneous networks. LISS, ISS, MISS, and JS show better performance in the network with 200 nodes in Fig. 9 than in the network with 600 nodes in Fig. 10, since the topology with 600 node has larger average delay.

Fig. 15 shows the initial 5 s throughput and the total throughput, as the average inter-arrival time of connec-



(a) Initial 5 second throughput.



(b) Total throughput.

Fig. 15. Server throughput with varying exponential inter-arrival time in 600-node topology.

tions between the server and clients ranges from 1 to 0.1 s. LISS and ISS perform better than TCP Reno in both cases but the gain is not too impressive. MISS and JS outperform others in terms of the initial 5 s throughput but show slightly worse performance in terms of the total throughput as the average inter-arrival time decreases.

The initial n second throughput of Fig. 16 illustrates how each model performs as n increases. The throughput gain of LISS and ISS in Fig. 16 is more noticeable with large n for the topology with 600 nodes, whereas relatively small n is enough to see the difference in Fig. 12 for the topology with 200 nodes. The reason that it takes larger n to observe throughput gain in case of 600-node topology is probably the longer average delay resulting in a slower self-clocking cycle. On the other hand, MISS and JS do not rely on ACKs to increase $cwnd$ initially. Consequently, they show good performance even with small n with significant decrease in terms of the total throughput due to the initial burst of packets. The throughput gain of LISS, ISS against TCP Reno is significant only when n is large in terms of the initial n second throughput with the large average link delay. Also, as the network size grows, fairness of TCP becomes harder to achieve due to heterogeneity of the environment, and $cwnd$ and $ssthresh$ determined by our methods tend to be inaccurate. Usually, MISS and JS start with large initial $cwnd$ values with a network with large delay bandwidth product. As a consequence, when the router queue size is



not large enough, MISS and JS may generate more packet drops with increasing network size.

5.3. Connections with short duration

In the previous subsections, the mean connection duration was set to 100 s for simulation. This workload models multimedia streaming rather than general web traffic, where a typical connection duration time is a few seconds. To evaluate our slow start strategies with connections with short duration, the mean duration time of connections between the server and clients is adjusted to 5 s following the Pareto distribution.

First, the simulation results in Fig. 17 is obtained using the same simulation parameters as Fig. 3 for the homogeneous network environment except that the mean connection duration is 5 s rather than 100 s. In this setting, ISS, LISS, MISS and JS all outperform TCP Reno in terms of the 5 s throughput and the total throughput. It is interesting to note that the effectiveness of MISS much deteriorated than other slow start strategies for short-lived connections.

Next, Fig. 18 shows simulation results using the topology with 200 nodes. Except for the shorter mean connection duration time, the parameters used in simulation of Fig. 11 remain the same. In this moderately heterogeneous environment, ISS, LISS, JS show better performance than TCP Reno in terms of the 5 s throughput and the total throughput. In particular, JS shows excellent performance in the 5 s throughput as in the case of long-lived connections. However, MISS performs worse than TCP Reno in terms of the 5 s throughput and the total throughput.

Finally, Fig. 19 illustrates the simulation results executed with the topology with 600 nodes. Simulation parameters are identical as executed in Fig. 14 except that the mean connection duration is changed from 100 s to 5 s. ISS and LISS show better performance than TCP Reno in most cases, while JS generates massive packet drops even though it shows excellent 5 s throughput. MISS does not perform well with connections with short duration both in terms of the 5 s throughput and the total throughput.

Overall, ISS, LISS and JS show similar behavior when the mean connection duration time changes. However,

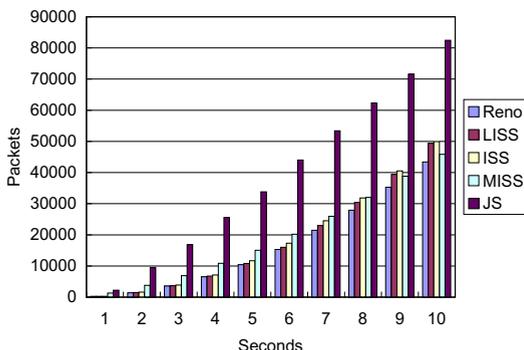


Fig. 16. Server n second throughput in 600-node topology.

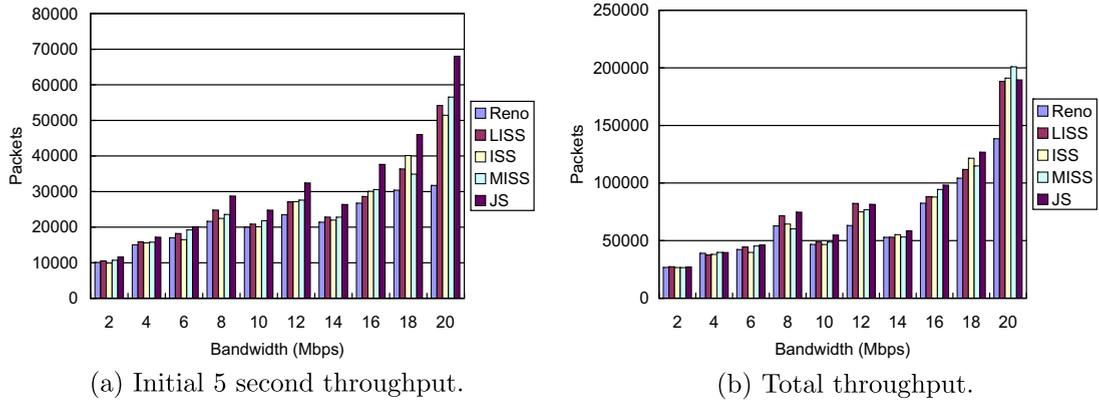


Fig. 17. Server throughput with varying bandwidth in homogeneous topology with short-lived connections.

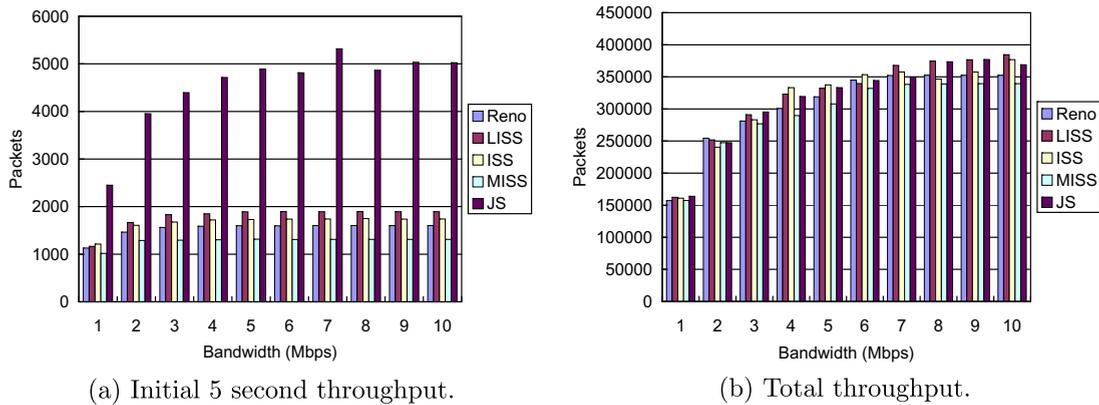


Fig. 18. Server throughput with varying bandwidth in 200-node topology with short-lived connections.

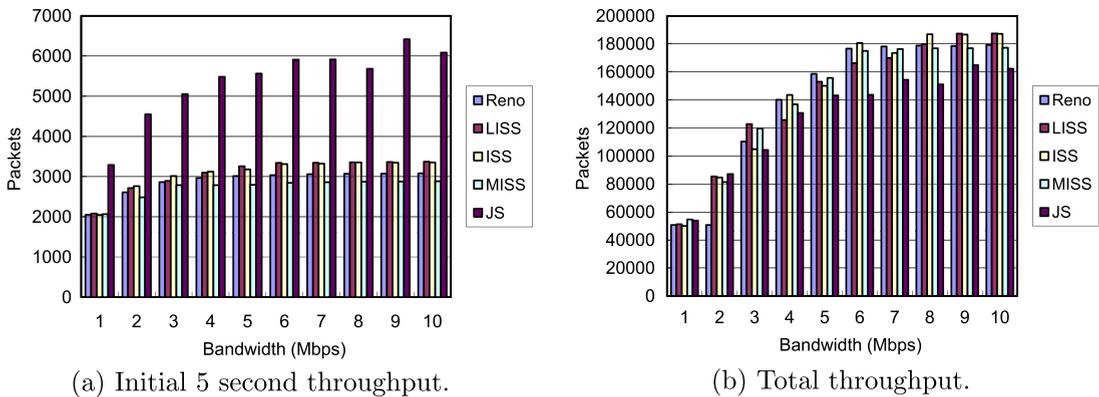


Fig. 19. Server throughput with varying bandwidth in 600-node topology with short-lived connections.

performance degradation of MISS for connections with short duration is evident. As MISS requires a long running TCP connection with occasional packet drops to calculate *cwnd* using Eq. (5), we conjecture that MISS is not able to obtain good *cwnd* values when the mean connection duration is short. Consequently, MISS may be effective only when the multimedia-like workload is considered.

5.4. Comparison with other methods

In this subsection, the performance of LISS is compared with modified versions of TCP Fast Start [7] and Zhang et al. [6]. For the purpose of reference, we call the first method the “cached” method and the second one the “averaged” method. With the cached method, when there

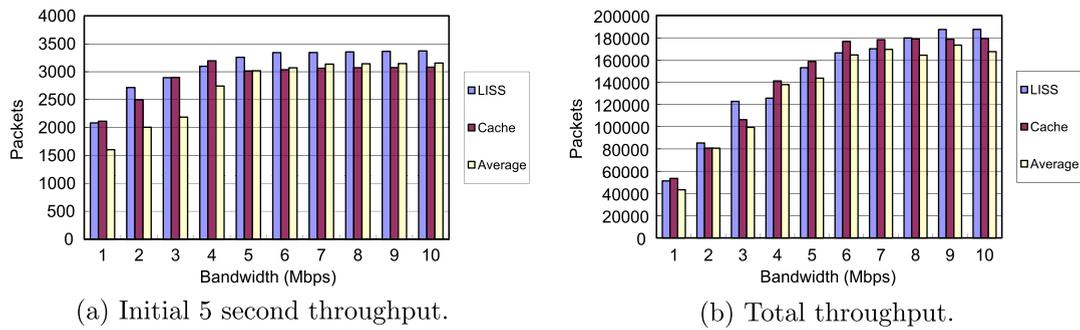


Fig. 20. Server throughput with varying bandwidth in 600-node topology for LISS, the cached method (labeled as cache) and the averaged method (labeled as average).

is an active connection sharing the same subnet with a new connection, the *ssthresh* value of the active connection is chosen for that of the new connection. Otherwise, when there is a cached *ssthresh* value having the same destination with a new connection, the cached value is used. If there is no cached value, the default *ssthresh* is used as described earlier in this section. For the averaged method, each subnet samples the *cwnd* value of each connection to the server and the number of the connections in the subnet, with the sampling interval of 1 s. The accumulated *cwnd* values are divided by the number of the connections sampled during the last 5 min to obtain the average *cwnd* value. This average value is updated every minute and then it is used as *ssthresh* for a new connection.

A typical simulation result of these three methods is shown in Fig. 20, where the simulation parameters are the same as in Fig. 19. The cached method is comparable to LISS, whereas the averaged method shows worse performance than the other two. With different simulation scenarios, it was found that the performance of the cached method improved when cached *ssthresh* values were recent, while LISS outperformed when there was a good chance to find connections sharing the same bottlenecks with a new connection. For example, when connections were long-lived ones, LISS showed the best performance, whereas the cached method showed good performance when it could find recent cache values with short-lived connections. The averaged method consistently underperformed throughout simulations, and this may be attributable to its low sampling rate and averaging rate. However, increasing those rates would imply more overhead.

From the observation above, in comparison with LISS, the cached method can be beneficial when a new connection has no other connection sharing the same bottlenecks. Naturally, LISS also can benefit from adopting the cached *ssthresh* values when these values are not too old. However, it is not clear how old is not too old as will be discussed in the next section.

6. Discussion

A few implementations of the TCP protocol do not reset the *ssthresh* value for at least several minutes when the TCP connection is idle [18]. This is based on the assumption

that the network conditions might be steady at least for several minutes [36,37]. The same assumption can be applied to connections terminated a few minutes earlier. If we can find the bandwidth of recently terminated connections of the same subnet, an estimate of a new connection bandwidth can be obtained. But how about the connections terminated more than a few minutes earlier? To answer this question it is necessary to measure extensive Internet traffic dynamics for long durations that is not in the scope of this study.

Another issue that should be considered is a method to identify subnets with common bottlenecks. The Autonomous System (AS) prefix table from Border Gateway Protocol (BGP) can be used in locating web client clusters [38]. If the AS prefix table is not available, Internet tomography can be considered [39]. However, many probe packets are necessary to get a good estimate using this technique. FlowMate [40] shows that it is possible to find connections with shared bottleneck links in the OS kernel effectively. For simplicity, we may take the prefix of subnet mask /24 assuming that most subnets have the subnet mask of /24. In fact, IP addresses with prefixes longer than /24 show strong geographical locality as found in Freedman et al. [41]. Also, it is reported in Cherkasova and Gupta [42] that there is temporal and geographic locality of media requests from clients to a server, increasing the chance to find connections with shared bottlenecks within a reasonable time window.

To get a better *cwnd* value, Eq. (5) can be refined. One way is to use L and S/w_0 from Eq. (2) in Eq. (5). As Eq. (5) does not work well with short-lived connections, this can be of help. For another, if $\tau_l - (t_c - t_l) < R$ in Eq. (5), it may imply that competing connections fill up the bottleneck earlier than when the next RTT terminates. In this case, packet pacing during the next RTT could be too aggressive and we can try to transmit

$$\frac{\tau_l - (t_c - t_l)}{R} \cdot cwnd,$$

packets in $\tau_l - (t_c - t_l)$. Unfortunately, the performance of this scheme was not satisfactory. It seems Eq. (5) can be used only as a coarse predictor of available bandwidth in the bottleneck.

7. Conclusion

We introduced LISS, ISS, MISS, and JS as TCP initial start-up models. LISS and ISS are the same as the original TCP slow start algorithm except for *ssthresh* calculation. JS is analogous to Zhang et al. [6], while the latter relies on a network monitor to obtain *ssthresh*.

Each model determines an initial *ssthresh* value of a new TCP connection. In addition, MISS tries to estimate a non-intrusive initial *cwnd* value to reduce the impact on the router, while JS uses *ssthresh* as the initial *cwnd*. MISS and JS also use packet pacing during slow start.

By way of simulation, the initial 5 s throughput and the total throughput of our schemes are compared against TCP Reno. The fast ramp-up of *cwnd* up to the *ssthresh* value helps to achieve higher throughput, when the router is equipped with reasonable queue size in a homogeneous network where fairness of TCP is easier to observe. In contrast, when a network is large and heterogeneous, the total throughput of our schemes tends to be lower than that of TCP Reno, as it becomes harder to determine good *cwnd* and *ssthresh* values. More importantly, as the initial *cwnd* and *ssthresh* values are large with large delay bandwidth product paths, packet drops are more likely to occur in a router with a small queue. It is also evident that the initial TCP start-up performance depends on *cwnd* increment policy. Even though it was not shown in the simulation, JS without packet pacing is prohibitive as it generates massive packet drops with large networks. In fact, JS generates sizeable packet drops even when packet pacing is engaged on the large delay bandwidth product path. However, it should be noted that packet drops are significantly reduced if routers have large queueing buffers for the large delay bandwidth product paths, achieving almost the same total throughput of ISS and LISS.

In conclusion, we believe MISS can work well within an ISP equipped with multimedia servers, where the average delay between clients and servers is small without requiring that routers have large buffers. In practice, Content Distribution Networks (CDN) like Akamai [43] use a similar architecture by placing dedicated (cache) servers at strategic locations nearby clients. In such cases, MISS can provide multimedia content to users with much less delay than TCP Reno without sacrificing the total throughput.

In situations where subscribers in large and heterogeneous networks with short-lived sessions should be considered, we believe LISS is a good choice. LISS does not require modification of the TCP algorithm and improves the TCP start-up performance even with routers with a limited buffer size. Simply, with LISS, *cwnd* of the oldest connection can be used to obtain *ssthresh* of a new TCP connection.

References

- [1] W.R. Stevens, TCP/IP Illustrated: The Protocols, vol. 1, Addison-Wesley Professional, 1993.
- [2] D.-M. Chiu, R. Jain, Analysis of the increase and decrease algorithms for congestion avoidance in computer networks, *Comput. Netw. ISDN Syst.* 17 (1989) 1–14.
- [3] V. Jacobson, Congestion avoidance and control, in: ACM SIGCOMM '88: Symposium Proceedings on Communications Architectures and Protocols, pp. 314–329.
- [4] ISI/USC, The network simulator – ns-2, 2008. [cited 2008 November 1]. Available from: <<http://www.isi.edu/nsnam/ns/>>.
- [5] J. Hoe, Improving the start-up behaviour of a congestion control scheme for tcp, in: ACM SIGCOMM '96: Conference Proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications, ACM, New York, NY, USA, 1996, pp. 270–280.
- [6] Y. Zhang, L. Qiu, S. Keshav, Speeding up short data transfers: Theory, architectural support, and simulation results, in: NOSSDAV '00.
- [7] V.N. Padmanabhan, R.H. Katz, Tcp fast start: a technique for speeding up web transfers, in: IEEE GLOBECOM '98, pp. 41–46.
- [8] M. Aron, P. Druschel, TCP: Improving Startup Dynamics by Adaptive Timers and Congestion Control, Technical Report, Rice University, 1998.
- [9] J.-C. Bolot, End-to-end packet delay and loss behavior in the Internet, *ACM SIGCOMM Comput. Commun. Rev.* 23 (1993) 289–298.
- [10] N. Hu, P. Steenkiste, Evaluation and characterization of available bandwidth probing techniques, *IEEE J. Selected Area Commun.* 21 (2003) 879–894.
- [11] S. Ryoung Kang, X. Liu, M. Dai, D. Loguinov, Packet-pair bandwidth estimation: stochastic analysis of a single congested node, *IEEE International Conference on Network Protocols*, vol. 0, IEEE Computer Society, 2004, pp. 1–5.
- [12] S. Keshav, A control-theoretic approach to flow control, *ACM SIGCOMM Comput. Commun. Rev.* 21 (1991) 3–15.
- [13] J. Strauss, D. Katabi, F. Kaashoek, A measurement study of available bandwidth estimation tools, in: Proceedings of the Third ACM SIGCOMM Conference on Internet Measurement, ACM, New York, NY, USA, 2003, pp. 39–44.
- [14] M. Allman, Improving TCP Performance Over Satellite Channels, Master's thesis, Ohio University, 1997.
- [15] K. Fall, S. Floyd, Simulation-based comparisons of Tahoe, Reno and Sack TCP, *ACM SIGCOMM Comput. Commun. Rev.* 26 (1996) 5–21.
- [16] M. Allman, S. Floyd, C. Partridge, RFC 3390: Increasing TCP's initial window, 2002.
- [17] R. Wang, G. Pau, K. Yamada, M.Y. Sanadidi, M. Gerla, Tcp startup performance in large bandwidth delay networks, in: Proceedings of IEEE INFOCOM '04, vol. 2, pp. 796–805.
- [18] V. Visweswaraiyah, J. Heidemann, Improving Restart of Idle TCP Connections, Technical Report USC TR 97-661, USC, 1997.
- [19] M. Allman, Tcp byte counting refinements, *ACM SIGCOMM Comput. Commun. Rev.* 29 (1999) 14–22.
- [20] C. Dovrolis, P. Ramanathan, D. Moore, What do packet dispersion techniques measure? in: Proceedings of IEEE INFOCOM '01, vol. 2, pp. 905–914.
- [21] B. Melander, M. Bjorkman, P. Gunningberg, A new end-to-end probing and analysis method for estimating bandwidth bottlenecks, in: IEEE GLOBECOM '00, vol. 1, pp. 415–420.
- [22] V. Paxson, End-to-end Internet packet dynamics, Proceedings of the ACM SIGCOMM '97 conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, vol. 27, ACM, 1997, pp. 139–154.
- [23] N. Hu, P. Steenkiste, Improving tcp startup performance using active measurements: algorithm and evaluation, in: IEEE ICNP '03: Proceedings of the 11th IEEE International Conference on Network Protocols, IEEE Computer Society, Washington, DC, USA, 2003.
- [24] L.S. Brakmo, L.L. Peterson, TCP Vegas: end to end congestion avoidance on a global Internet, *IEEE J. Selected Areas Commun.* 13 (1995) 1465–1480.
- [25] M. Jain, C. Dovrolis, End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput, in: ACM SIGCOMM '02: Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, ACM, New York, NY, USA, 2002, pp. 295–308.
- [26] R. Riedi, J. Navratil, R. Baraniuk, L. Cottrell, pathchirp: Efficient available bandwidth estimation for network, in: Passive and Active Measurement Workshop '03.
- [27] S. Savage, N. Cardwell, T. Anderson, The case for informed transport protocols, in: HOTOS '99: Proceedings of the Seventh Workshop on Hot Topics in Operating Systems, IEEE Computer Society, Washington, DC, USA, 1999, pp. 58–59.
- [28] H. Balakrishnan, H.S. Rahul, S. Seshan, An integrated congestion management architecture for internet hosts, *SIGCOMM Comput. Commun. Rev.* 29 (1999) 175–187.
- [29] P.R. Selvidge, B.S. Chaparro, G.T. Bender, The world wide wait: effects of delays on user performance, *Int. J. Indust. Ergonom.* 29 (2002) 15–20.
- [30] E. Wang, Human-Computer Network Interaction: Delay Effects and its Mediators, Ph.D. Thesis, Purdue University, 2005.

- [31] J.F. Kurose, K.W. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*, third ed., Addison-Wesley, 2002.
- [32] A. Mena, J. Heidemann, An empirical study of real audio traffic, in: *Proceedings of IEEE INFOCOM '00*, vol. 1, pp. 101–110.
- [33] K. Park, G. Kim, M. Crovella, On the relationship between file sizes, transport protocols, and self-similar network traffic, Technical Report BU-CS-96-016, Computer Science Department, Boston University, 1996.
- [34] E. Veloso, V. Almeida, J. Wagner Meira, A. Bestavros, S. Jin, A hierarchical characterization of a live streaming media workload, *IEEE/ACM Trans. Netw.* 14 (2006) 133–146.
- [35] K.L. Calvert, M.B. Doar, A. Nexion, E.W. Zegura, G. Tech, G. Tech, Modeling internet topology, *IEEE Commun. Magaz.* 35 (1997) 160–163.
- [36] V. Paxson, End-to-end routing behavior in the Internet, in: *ACM SIGCOMM '96: Conference Proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ACM, New York, NY, USA, 1996, pp. 25–38.
- [37] Y. Zhang, N. Duffield, V. Paxson, S. Shenker, The constancy of internet path properties, in: *IMW '01: Proceedings of the First ACM SIGCOMM Workshop on Internet Measurement*, ACM, New York, NY, USA, 2001, pp. 197–211.
- [38] B. Krishnamurthy, J. Wang, On network-aware clustering of web clients, in: *ACM SIGCOMM '00: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ACM, New York, NY, USA, 2000, pp. 97–110.
- [39] M. Coates, A. Hero, R. Nowak, B. Yu, Internet tomography, *Signal Process. Magaz.* 19 (2002).
- [40] O. Younis, S. Fahmy, Flowmate: scalable on-line flow clustering, *IEEE/ACM Trans. Netw.* 13 (2005) 288–301.
- [41] M. Freedman, M. Vutukuru, N. Feamster, H. Balakrishnan, Geographic locality of ip prefixes, in: *Internet Measurement Conference (IMC)*, USENIX Association, 2005, pp. 13–13.
- [42] L. Cherkasova, M. Gupta, Analysis of enterprise media server workloads: access patterns, locality, content evolution, and rates of change, *IEEE/ACM Trans. Netw.* 12 (2004) 781–794.
- [43] L. Kontothanassis, R. Sitaraman, J. Wein, D. Hong, R. Kleinberg, B. Mancuso, D. Shaw, D. Stodolsky, A transport layer for live streaming in a content delivery network, *Proc. IEEE* 92 (2004) 1408–1419.



InKwan Yu received his B.A. degree in philosophy and M.S degree in computer science from Korea University, Seoul, South Korea. In 2003, he came to the Department of CISE, University of Florida, Gainesville, Florida, USA. He graduated from University of Florida in 2009 with his Ph.D. degree in computer engineering. InKwan Yu also worked at Korea Institute for Defense Analysis and Softforum in South Korea.



Richard Newman is an Assistant Professor of Computer & Information Science & Engineering at the University of Florida. He earned his Ph.D. in Computer Science from University of Rochester in 1986. His research is primarily in distributed systems, computer networking and security. His industry- and government-sponsored projects on these topics has brought in over \$3.5 million and lead to over 100 refereed technical publications. Projects have included design and analysis of network protocols, particularly for powerline communications; mechanisms for efficiently satisfying QoS requirements on communication channels; design and development of distributed conferencing systems; usable security; and analysis of network covert channels, anonymity, and other information hiding techniques. Methods by which independent users and systems can work together securely to achieve a common goal is a special interest.