# On Queue Length and Link Buffer Size Estimation in 3G/4G Mobile Data Networks

Stanley C. F. Chan, K. M. Chan, K. Liu, and Jack Y. B. Lee

**Abstract**—The emerging mobile data networks fueled by the world-wide deployment of 3G, HSPA, and LTE networks created new challenges for the development of Internet applications. Unlike their wired counterpart, mobile data networks are known to exhibit highly variable bandwidth. Moreover, base stations are often equipped with large buffers to absorb bandwidth fluctuations to prevent unnecessary packet losses. Consequently to optimize protocol performance in mobile data networks it is essential to be able to accurately characterize two key network properties: queue length and buffer size of the bottleneck link. This work tackles the challenge in estimating these two network properties in modern mobile data networks. Using extensive trace-driven simulations based on actual bandwidth trace data measured from production mobile data networks, we show that existing queue-length and link buffer size estimation algorithms no longer work well in bandwidth-varying networks. We develop a novel sum-of-delays algorithm which incorporates the effect of bandwidth variations into its estimation. Extensive trace-driven simulation results show that it can accurately estimate the queue length and link buffer size under both fixed and varying bandwidth conditions, outperforming existing algorithms by up to two orders of magnitude.

**Index Terms**—Queue length, link buffer size, estimation algorithm, sum-of-delays, mobile data networks, passive and active estimations

## 1 Introduction

It is well-known that the performance of Internet transport and application protocols depends heavily on the characteristics of the underlying network links, and in particular, the bottleneck link. Apart from the impact of competing traffic, a bottleneck link has three primary parameters, namely link bandwidth, buffer size, and queue length. Many previous works have investigated methods to estimate these link parameters, for example link bandwidth estimation [1]–[3], link buffer size estimation [4]–[6], and queue length estimation [7]–[9].

A common assumption among these previous works is that the link bandwidth is constant, which is largely valid for wired networks as the physical link typically has a fixed transmission data rate. However with the rapidly emerging mobile data networks such as 3G [10], HSPA [11], and LTE [12], this assumption is no longer valid.

To illustrate, Fig. 1 plots the measured bandwidth of a production 3G network over a time interval of 10 seconds where there is only a single UDP data flow going from a wired server to a 3G-connected receiver. It is evident that unlike fixed network, mobile data network can exhibit significant bandwidth fluctuations over a short timescale (also over longer timescales, not shown here) due to fluctuations in radio conditions, interference from other wireless devices, contention among multiple users in the same cell, device mobility, etc. In addition, modern mobile networks typically implement link-layer retransmission to recover frame losses. Consequently a frame loss will translate into longer transmission time, resulting in a sudden bandwidth drop as observed by the receiver. These pose a new challenge to existing link property estimation algorithms as the assumption of fixed link bandwidth clearly no longer holds.

This work tackles the above-mentioned challenge by developing a novel *sum-of-delays* (SoD) algorithm to estimate link buffer size of the network and queue length of the underlying network connection. In contrast to previous works, SoD does not assume network bandwidth to be constant (as in [4]–[6]), or rely on indirect bandwidth estimation via TCP's congestion window (as in [7]–[9]). Instead, SoD *measures* the *actual* bandwidth from ACK timings and keeps track of the sequence of *past* measurements so that it can *compensate* for link bandwidth variations in estimating link buffer size and queue length.

SoD can be implemented using both *active estimation* – commonly employed in network measurement tools, and *passive estimation* – which eliminates the need for sending extra measurement packets and simply measures the in-band data packet timings to perform estimation. The latter approach enables the integration of the estimation algorithm into existing transport protocols such as TCP to optimize flow and congestion controls.

Through extensive trace-driven simulations using actual bandwidth traces captured from production 3G networks we show that the proposed estimation algorithm outperforms existing link buffer size estimation algorithms such as max-min [4]–[6] and loss-pair [6] by multiple

• *The authors are with the Department of Information Engineering, Chinese University of Hong Kong, Hong Kong.*
*E-mail: {ccf008, ckm011, lk008, yblee}@ie.cuhk.edu.hk.*

*Manuscript received 24 Apr. 2012; revised 31 Aug. 2013; accepted 27 Sep. 2013. Date of publication 3 Oct. 2013; date of current version 29 May 2014. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.*
*Digital Object Identifier 10.1109/TMC.2013.127*

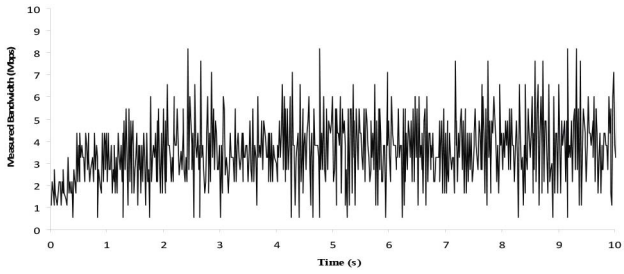



Fig. 1. Measured bandwidth of a production 3G network.

orders of magnitude in terms of estimation errors and also achieve better accuracy than existing queue length estimation algorithms like the Vegas method [7] and the FAST method [8] in delay-based TCP variants.

An accurate estimation of the link buffer size and queue length for a traffic flow has a number of significances to mobile communications. First, transport protocols such as TCP were originally designed for wired networks of which the link buffer size is shared and is often small (e.g., in the order of tens of kilobytes per port). This is no longer the case in mobile data networks where link buffer at the base station is allocated on a per-user basis with buffer size often in hundreds of kilobytes, causing TCP to perform suboptimally [13]. Therefore knowledge of the link buffer size and queue length will enable the development of a new generation of TCP variants which take mobile networks' large link buffer size into account to optimize throughput performance in mobile data networks.

On the other hand, queue length has a direct impact to data delivery delay. Thus for real-time applications such as live video streaming (e.g., news, sports, and concerts), voice-over-IP, and audio/video conferencing, knowledge of the queue length can be exploited to (a) design new rate-adaptation algorithms to balance throughput and delay for better quality of experience; or (b) design mobile network resource allocation algorithms to allocate bandwidth resources not just based on bandwidth demand, but also based on delay constraints as well.

Second, many application protocols employ feedbacks from receivers to regulate the traffic flow. The existence of a large link buffer may introduce substantial delay to the feedbacks and potentially cause stability problems in closed-loop control protocols. Again knowledge of the link buffer size will enable these protocols to compensate for the extra delays to ensure stability over mobile networks.

Third, the ability to estimate queue length at the bottleneck link opens up a new way for mobile operators to monitor the performance of their subscribers and cell sites. Due to mobile network's inherent bandwidth fluctuations, link/transport layer throughput may not be a good metric to reflect the utilization and congestion experienced by mobile subscribers. By passively monitoring the queue lengths of on-going or past traffic flows, the network operator can then obtain a more realistic picture of the network's utilization and the service quality perceived by their subscribers.

The rest of the paper is organized as follows: Section 2 re-examines existing algorithms for estimating link buffer size and queue length, and evaluates their accuracies when applied to mobile data networks; Section 3 presents the proposed sum-of-delays estimation algorithm. Section 4 evaluates the sum-of-delays algorithm and compares it to existing algorithms; Section 5 summarizes the study and outlines some future work.

TABLE 1
Summary of Notations

| Symbol | Description |
|---|---|
| $P_{d_1}$ | Propagation delay from sender to the link buffer |
| $P_{d_2}$ | Propagation delay from link buffer to the receiver |
| $P$ | Total downlink (from sender to receiver) propagation delay |
| $U$ | Uplink (from receiver to sender) delay |
| $q_i$ | Queueing delay of the $i^{\text{th}}$ packet |
| $q_{max}$ | Maximum queueing delay |
| $n_i$ | Number of packets queued-up at the link buffer when $i^{\text{th}}$ packet enters the buffer |
| $S$ | Packet size |
| $L$ | Link buffer size (including the one in service) |
| $T$ | Transmission delay of the packet |
| $C$ | Measured link capacity |
| $cwnd_i$ | Congestion window size of the $i^{\text{th}}$ packet |
| $rtt_i$ | Round-trip time of the $i^{\text{th}}$ packet |

## 2 Background and Related Works

In this section we re-examine two well-known algorithms on link buffer size estimations, namely max-min [5], [6] and loss-pair [6], and the queue length estimation algorithms used in TCP Vegas [7], FAST TCP [8], and TCP Veno [9]

We consider the system model depicted in Fig. 2 where a single sender transmits packets via a bottleneck link to a receiver. None of the bottleneck link's parameters are known *a priori* and the propagation delays are constant.

The goal is to estimate the queue length continuously, denoted by $q_i$, and link buffer size, denoted by $L$, at the sender via observation of the acknowledgement packet arrival times. Table 1 summarizes the notations used in the following discussions.

### 2.1 Link Buffer Size Estimation

Liu and Crovella [6] proposed the max-min method to estimate the link buffer size from the estimated transmission delay and the differences between maximum and minimum round-trip times (RTTs). Claypool *et al.* [5] further incorporated the max-min method into a measurement tool for use in access networks.

The principle of both max-min [5], [6] and loss-pair [6] methods is to use RTT as a mean to measure the link buffer size. Specifically, let $P = P_{d1} + P_{d2}$ be the total downlink (from sender to receiver) propagation delay, $T$ be the transmission delay for one packet, and $U$ be the uplink (from receiver to sender) delay, which are all constant. Next let $q_i$ be the queueing delay experienced by packet $i$ at the bottleneck link, and $n_i$ be the number of packets already queued-up at the bottleneck link buffer upon its arrival at the queue. Then we can compute the RTT for packet $i$, denoted by $rtt_i$, from

$$rtt_i = P + T + U + q_i. \quad (1)$$

In a fixed-bandwidth link, the queueing delay of packet $i$ is simply equal to the total time to de-queue the $n_i$ packets already in the queue plus the residual transmission time, denoted by $\delta$, for the packet already in service, i.e.,

$$q_i = n_i T + \delta. \tag{2}$$

Given the link buffer size $L$, we have $n_i \leq L-1$. Also the residual transmission time is bounded by $\delta \leq T$. Therefore the maximum queueing delay, denoted by $q_{max}$, is given by

$$\begin{aligned} q_{\max} &= \max\{(L-1)T + \delta\} \\ &= LT. \end{aligned} \tag{3}$$

The minimum queueing delay is simply equal to zero. As the propagation and transmission delays are constant we can then compute the maximum and minimum RTTs from

$$rtt_{\max} = P + T + U + q_{\max} \tag{4}$$

$$rtt_{\min} = P + T + U. \tag{5}$$

Substituting (3) and (5) into (4), we have

$$rtt_{\max} = rtt_{\min} + LT. \tag{6}$$

Similarly, substituting (2) and (5) into (1), we have

$$rtt_i = rtt_{\min} + n_i T + \delta. \tag{7}$$

Rearranging terms in (6), we can determine the link buffer size from [4],

$$L = \frac{rtt_{\max} - rtt_{\min}}{T}. \tag{8}$$

With knowledge of measured link capacity ($C$) and known packet size ($S$), we could compute the estimated transmission delay ($T$) from

$$T = S/C \tag{9}$$

and determine the link buffer size accordingly:

$$L = \frac{(rtt_{\max} - rtt_{\min})C}{S}. \tag{10}$$

The max-min method [5], [6] is designed for active estimation where the sender sends a series of measurement packets through the bottleneck link at a rate higher than the link bandwidth to induce queue overflow. The receiver returns an acknowledgement (ACK) packet to the sender for every packet received. The sender can then measure the minimum RTT $rtt_{min}$, maximum RTT $rtt_{max}$, and link bandwidth $C$ from the ACK packet arrival times. With $S$ known the sender can compute $L$ using (10). The estimation process can also be done by the receiver and the interested readers are referred to the work by Hirabaru [4] for more details.

From (10) we can see that the accuracy of the max-min method hinges upon the accuracy in measuring the three parameters, namely $rtt_{max}$, $rtt_{min}$, and $C$. In particular, if there is other cross traffic sharing the same bottleneck link, the RTT measurements will be modulated by the competing flows.

Liu and Crovella [6] tackled this problem in their improved loss-pair method. First, they only capture the RTTs of the two packets just before and after a loss event. This filters out samples not related to buffer overflow at the bottleneck link. Second, they analyze the distribution of the samples to determine their mathematical mode to further filter out noises due to cross traffic. These two techniques improved the accuracy of the loss-pair method.

### 2.2 Queue Length Estimation

To the best of the authors' knowledge there is no known measurement tool designed solely for queue length estimation purpose. The reason being that unlike link buffer size, which is a physical network property, queue length can vary from time to time depending on many parameters, including offered traffic load, traffic characteristic, link capacity, and so on. Therefore queue length measurement is meaningful only in the context of the actual data flow generated by the transport and/or application protocols.

Some TCP variants do implement algorithms to either implicitly or explicitly estimate the queue length at the bottleneck link for congestion control purpose. One well known example is TCP Vegas [7] which employs an algorithm to estimate the queue length from the congestion window size and the difference between *current* and *minimum* round-trip times (RTTs). Similar algorithms have also been adopted in FAST TCP [8] and TCP Veno [9].

These queue-length estimation algorithms are inherently passive in that only the actual data and ACK packet timings are used for estimating the queue length. No extra measurement packets are generated in the process.

During operation, the TCP sender continuously measures the RTT, denoted by $rtt_i$, and records the congestion window size, denoted by $cwnd_i$, at the time ACK packet $i$ is received. It then keeps track of the minimum $rtt_{min}$ by

$$rtt_{\min} = \min\{rtt_i | \forall i\} \tag{11}$$

and then computes the estimated queue length, denoted by $n'_i$, from [7]:

$$n'_i = \frac{cwnd_i}{rtt_i}(rtt_i - rtt_{\min}), \tag{12}$$

where $cwnd_i/rtt_i$ is the estimated bandwidth and $(rtt_i - rtt_{min})$ is the queueing time. Hence their product is the estimated queue length.

One shortcoming of the Vegas method is that it uses the current congestion window size to estimate the queue length in the previous RTT. In case the congestion window size changes significantly from the last RTT the estimation accuracy will suffer. This is why TCP Vegas only performs queue length estimation during the congestion avoidance phase where the congestion window size changes slowly.

FAST TCP [8] addressed this shortcoming by keeping track of the past congestion windows, and using the one at the time of the original data transmission instant to compute (12). This improves accuracy and also enables FAST TCP to estimate queue length during both slow-start and congestion avoidance phases. Nevertheless, neither the Vegas nor the FAST algorithm can be applied in TCP's loss recovery phase as the congestion window size by then no longer correlates with the network bandwidth available.

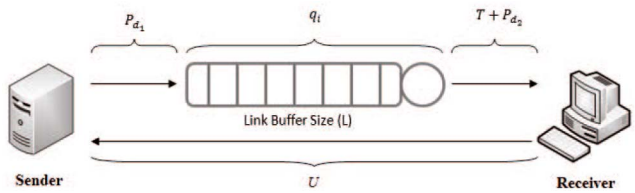



Fig. 2. System model for bottleneck link buffer size estimation.

### 2.3 Performance over Fixed Networks

In this section we re-examine the performance of existing link buffer size estimation algorithms and queue-length estimation algorithms using a simulator implemented in NS2 [14]. In this section we first evaluate them in their intended network model where the bottleneck-link bandwidth is fixed.

*Link buffer size estimation:* We implemented the max-min and loss-pair algorithms using UDP as the transport protocol using the network model as depicted in Fig. 2. The simulation parameters in Table 2 are set according to typical values in production 3G networks. We did conduct additional simulations with different delay parameters and found the results to be consistent. Note also that we did not introduce packet loss into the simulations for two reasons. First, max-min and loss-pair are inherently insensitive to packet loss as they primarily rely on measuring the maximum and minimum RTT over large number of samples. Second, the Vegas/FAST algorithms were not designed to operate in TCP's loss recovery phase so introducing random packet loss will have no impact to their estimation process as it is not performed during TCP's loss recovery phase at all. Thus for consistency we employ zero packet loss rate in all simulations.

We define two performance metrics called *absolute* and *relative* link buffer size estimation errors, denoted by $E_A$, $E_R$ respectively to evaluate the algorithms' performance:

$$E_A = \text{estimated link buffer size} - \text{actual link buffer size} \quad (13)$$

$$E_R = \frac{E_A}{\text{actual link buffer size}} \times 100\%. \quad (14)$$

Fig. 3 plots the estimation errors for max-min and loss-pair in active estimation for bottleneck link with buffer size ranging from 100 to 1000 packets. Not surprisingly the algorithms perform well in fixed-bandwidth networks. Their

TABLE 2
Summary of Simulation Parameters

| Parameters | Value | Description |
|---|---|---|
| $P_{d_1}$ | 10ms | Propagation delay from sender to bottleneck link |
| $P_{d_2}$ | 70ms | Propagation delay from bottleneck link to receiver |
| $U$ | 80ms | Propagation delay from receiver to sender |
| $L$ | 100 to 1000 | Bottleneck link buffer size |
| $C$ | 0.8 - 7.2 Mbps | For fixed bandwidth case |
| $T$ | 100s | Simulation duration |
| *# of runs* | 10 | Number of simulation runs averaged per data point |
| *Loss rate* | 0% | Packet loss rate |

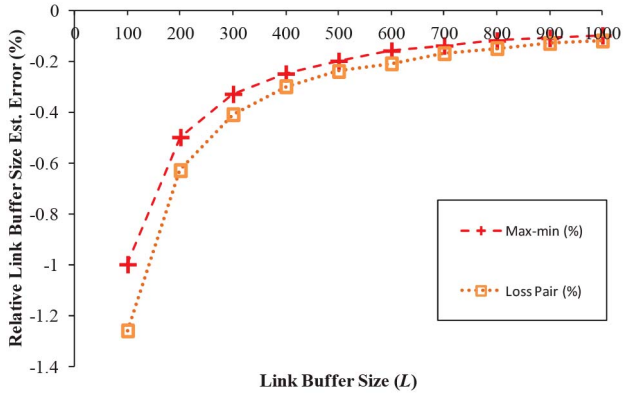



Fig. 3. Relative link buffer size estimation errors for fixed bottleneck link bandwidth (link buffer size, relative error, fixed bandwidth).

accuracies also improve as link buffer size increases (i.e., due to increases in the denominator of (14)).

*Queue-length estimation*: We implemented two queue-length estimation algorithms, namely the Vegas algorithm in TCP Vegas [7] and the FAST algorithm in FAST TCP [8], both using passive estimation. In addition, we also implemented the Vegas algorithm over TCP CUBIC [16]. In this case the TCP flow operates according to TCP CUBIC but the RTTs and congestion window size measurements are then fed into the Vegas algorithm to compute the estimated queue length. This special combination enables us to evaluate the performance of the Vegas estimation algorithm over one of the most widely deployed TCP variants in the Internet.

We define a relative queue length estimation error $Q_R$ and absolute queue length estimation error $Q_A$ as the performance metric:

$$Q_A = \text{estimated queue length} - \text{actual queue length} \quad (15)$$

$$Q_R = \frac{Q_A}{\text{average queue length}} \times 100\%. \quad (16)$$

Fig. 4(a) plots the estimation errors for the three cases for bottleneck link buffer size ranging from 100 to 1000 packets. There are two anomalies. First, FAST TCP achieved the best accuracy in all cases except the case with a link buffer size of 100 packets where the estimation error jumped to around 60%. This is due to a parameter setting in FAST TCP [8] which specifies the target number of inflight packets to maintain for the data flow. As the default value according to [8] is 100 packets it means FAST TCP will easily overflow the bottleneck link's buffer if the latter's size is close to or below the parameter value. As a result tuning of FAST TCP's parameter will have a significant impact on the performance of queue-length estimation.

Second, the results suggest that the Vegas algorithm when applied to a TCP Vegas flow in fact *underperformed* the same when applied to a TCP CUBIC flow. This is due to differences between TCP Vegas's and TCP CUBIC's congestion control algorithms. Specifically, TCP Vegas is designed to maintain a small number of packets (around 6 in our simulations) in the bottleneck link buffer. By contrast, TCP CUBIC is far more aggressive and in our simulations TCP CUBIC maintained an average queue length at around 90%

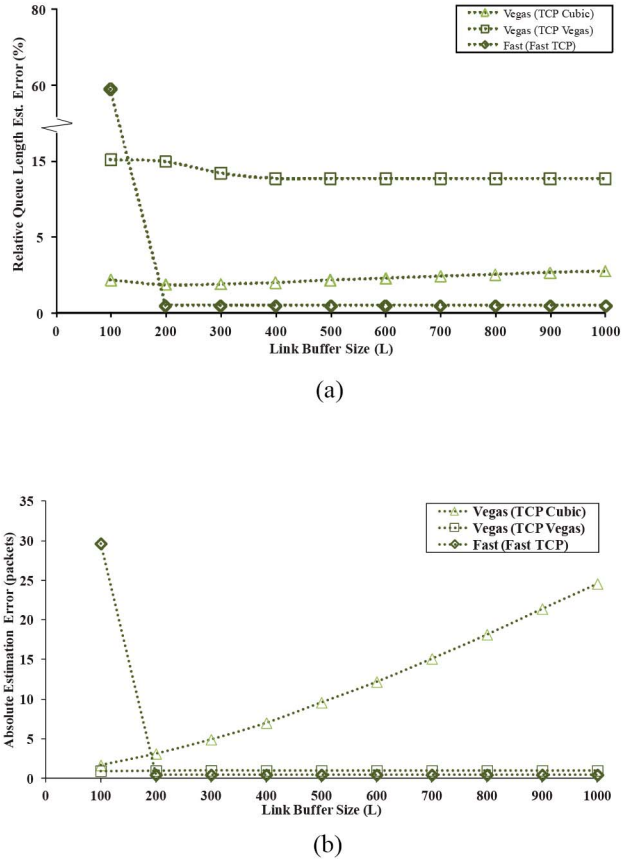



Fig. 4. (a) Relative queue length estimation errors, and (b) Absolute queue length deviation in different TCP variants for fixed bottleneck link bandwidth (queue length, relative error, fixed bandwidth).

of the link buffer size. Thus the larger denominator in (14) results in lower estimation errors for TCP CUBIC.

If we compare the estimation error in absolute number of packets as shown in Fig. 4(b) we can see that TCP CUBIC did exhibit higher absolute errors when compared to TCP Vegas and FAST TCP, especially for large link buffer sizes. This is because in TCP CUBIC, the congestion window grows faster for larger window sizes. Large link buffer size allows the congestion window to grow larger before buffer overflow occurs, thus causing higher estimation errors in computing (12) where the current congestion window size is used in place of the previous RTT's window size. FAST TCP does not suffer from this problem as it records the sequence of past window sizes and uses the correct one in computing (12), hence maintaining consistent estimation accuracy for a wide range of link buffer sizes.

### 2.4 Performance over Bandwidth-Varying Networks

In this section we apply the existing link buffer size and queue length estimation algorithms to mobile data networks. The only difference from the fixed network model in the previous section is that the link bandwidth $C$ is no longer constant, but time-varying subject to the effect of fading and signal interferences. To capture the actual bandwidth variations in production 3G/HSPA networks we conducted measurements by transmitting UDP datagrams from the sender to the receiver at a rate higher than

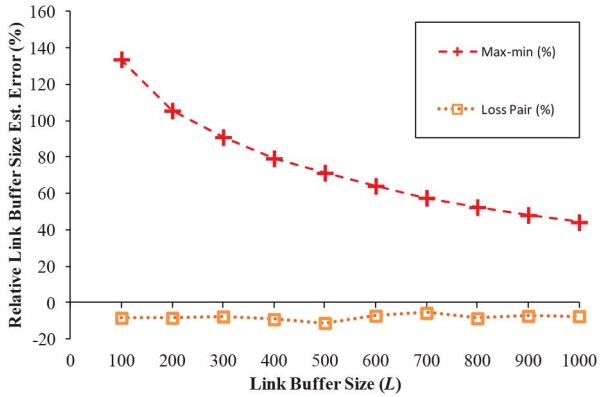



Fig. 5. Relative link buffer size estimation errors for varying bottleneck link bandwidth ($L$ is the actual link buffer size) (link buffer size, relative error, variable bandwidth).

the link bandwidth, and then recording the UDP datagram arrival timings at the receiver. As the link was flooded with packets all link bandwidth had been utilized and by counting the number of packet arrivals we obtained a time series of actual link bandwidth availability. It is worth noting that this time series data incorporated the properties of the entire link layer, including bandwidth/resource allocation algorithms, scheduling algorithms, and link-layer retransmissions.

In the simulator the radio link modulated its bandwidth according to this bandwidth time series data to replicate the bandwidth fluctuations. Each data point is the average of 10 simulations, each employing a different link bandwidth time series obtained from production 3G/HSPA networks.

We first evaluate the max-min and loss-pair link buffer size estimation algorithms in Fig. 5. It is clear that both algorithms exhibited significantly higher estimation errors compared to the fixed-bandwidth case.

Max-min in particular degraded significantly with estimation errors increasing from ~50% to ~130% as the link buffer size decreased. This is because the estimation algorithm cannot distinguish between queueing delay and variations in transmission time due to bandwidth variation. As smaller link buffer sizes result in shorter queueing delays, the relative impact of transmission time variations will increase as link buffer size decreases. It is worth noting that loss-pair degraded less compare to max-min because it does not base its estimation on the worst-case measurement, but determines the mathematical mode of the measured samples to filter out noises.

If we take a link buffer size of 300 packets for comparison, then the estimation errors for max-min and loss-pair increased from −0.3% to ~90%, and from −0.4% to ~10% respectively when moving from a fixed-bandwidth network to a mobile data network. It is clear from the results that the existing link buffer size algorithms, max-min and loss-pair, were designed for network with fixed bottleneck link bandwidth and consequently are not suitable for use in the emerging mobile data networks where bandwidth-varying radio link is the norm rather than the exception.

Next we re-examine queue length estimation algorithms, namely the Vegas estimation algorithm applied to TCP Vegas and TCP CUBIC, and the FAST algorithm in FAST

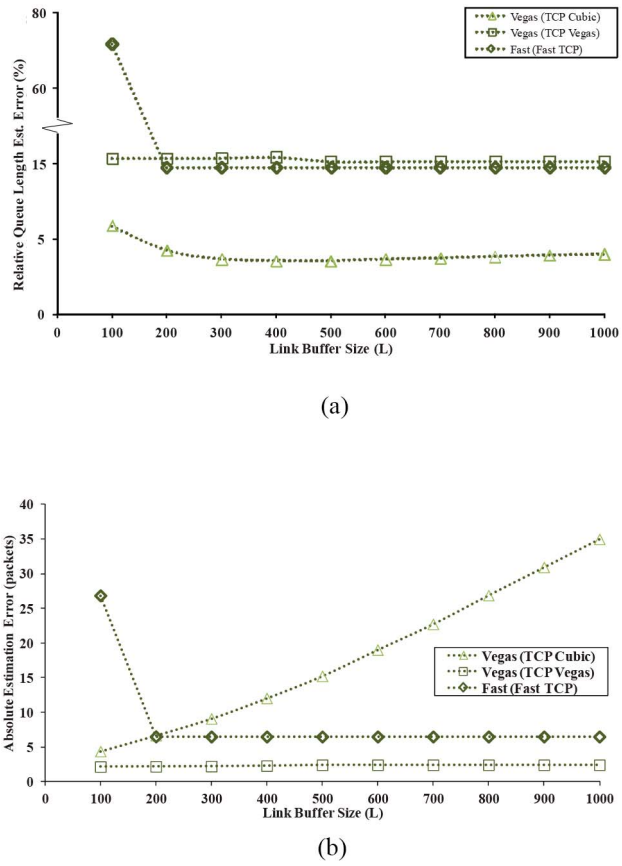



Fig. 6. (a) Relative queue length estimation errors, and (b) Absolute queue length deviation in different TCP variants for varying bottleneck link bandwidth (queue length, relative error, variable bandwidth).

TCP. Fig. 6(a) and (b) plot the percentage of estimation errors, and absolute estimation errors for the three cases. Comparing Fig. 6(a) with Fig. 4(a) we can see that when going from fixed-bandwidth network to varying-bandwidth network all three algorithms exhibited higher estimation errors, with FAST TCP degraded the most (from 0.5% to ~15%), follows by Vegas over TCP CUBIC (from ~3% to ~4%), and lastly Vegas over TCP Vegas (from ~14% to 15%).

Apart from the higher estimation errors, the algorithms are limited to queue length estimation during TCP's slow-start (Vegas and FAST) and congestion-avoidance (FAST) phases only. This is due to their reliance on the congestion window size as a mean to estimate the network bandwidth. When congestion is detected the congestion window will be cut drastically and thus can no longer serve as a reliable estimate of the network bandwidth. This could severely limit their application to some mobile data networks in case not all random packet losses can be successfully recovered by link-layer retransmissions. We tackle these challenges in the next section by developing a novel algorithm which does not rely on congestion window size in estimating the link buffer size and queue length.

## 3 Sum-of-Delays Method

The primary source of inaccuracies in the max-min and loss-pair is the assumption of constant network bandwidth.

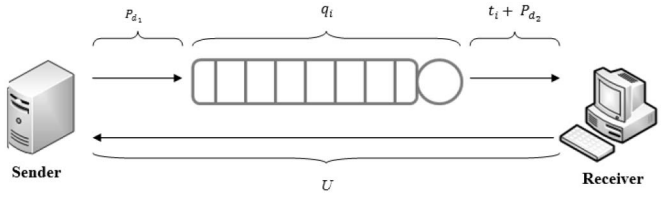



Fig. 7. Revised system model to incorporate the effect of bottleneck link bandwidth variations.

Even though the Vegas/FAST algorithms do indirectly estimate bandwidth via the TCP congestion window size, the estimation is nonetheless far from accurate. By contrast, we argue that instead of indirectly *estimating* the network bandwidth, it is in fact possible to directly *measure* the bandwidth from ACK packet timings. Moreover, by keeping track of the series of past bandwidth measurements, we can then *compensate* for the impact of network bandwidth variations in performing link buffer size and queue length estimations. We describe below a new sum-of-delays (SoD) method which incorporates this principle in link buffer size and queue length estimations.

Fig. 7 depicts the revised network model for a network with varying bottleneck link bandwidth. Note that variations in the link bandwidth will manifest as variations in packet transmission time. Specifically, the packet transmission time is equal to the packet size divided by the average bandwidth when the packet was transmitted by the bottleneck link. Let $t_i$ be the transmission time for packet $i$, then we can incorporate the effect of bandwidth variations through the series $\{t_i\}$, which can be computed from the measured ACK packet inter-arrival times for packets $i$ and $(i-1)$, denoted by $r_i$ and $r_{i-1}$ respectively:

$$t_i = r_i - r_{i-1} \tag{17}$$

provided that the link does not become idle during this time.

Consider the scenario when packet $i$ arrives at the bottleneck link to find that there are $n_i$ packets already queueing, plus packet $(i - n_i - 1)$ currently being transmitted. Then the queueing delay of packet $i$, denoted by $q_i$, is equal to the total time to de-queue all $n_i$ packets, i.e., the summation of transmission delays of packet $(i-1)$ to $(i-n_i)$, plus the residual time to transmit packet $(i - n_i - 1)$, denoted by $\delta_i$:

$$q_i = t_{i-1} + t_{i-2} + \cdots + t_{i-n_i} + \delta_i. \tag{18}$$

Note that the residual transmission time $\delta_i$ is bounded from above by the packet's transmission delay and bounded from below by zero, i.e.,

$$0 \le \delta_i \le t_{i-n_i-1} \tag{19}$$

Substituting the bounds of (19) into (18), we can then obtain the corresponding bounds for $q_i$:

$$t_{i-1} + t_{i-2} + \cdots + t_{i-n_i} \le q_i \le t_{i-1} + t_{i-2} + \cdots + t_{i-n_i} + t_{i-n_i-1}$$

$$\text{or} \qquad \sum_{k=i-n_i}^{i-1} t_k \le q_i \le \sum_{k=i-n_i-1}^{i-1} t_k. \tag{20}$$

Now our goal is to determine $n_i$, which is the queue length at the time packet $i$ arrives at the bottleneck link. In (20) the transmission delays $\{t_k\}$ can be computed from the measured ACK packet inter-arrival times using (17). The remaining unknown term is $q_i$.

Unfortunately the queueing time $q_i$ cannot be measured directly due to bandwidth variations. However it is possible to obtain a lower bound for $q_i$. Specifically, the RTT for packet $i$ as measured by the sender is equal to the sum of its queueing delay ($q_i$), transmission time ($t_i$) and propagation delays:

$$rtt_i = q_i + t_i + U + P_{d_1} + P_{d_2} \tag{21}$$

and the minimum RTT measured will be equal to

$$rtt_{\min} = \min\{q_i + t_i | \forall i\} + U + P_{d_1} + P_{d_2}. \tag{22}$$

Subtract (22) from (21) we can then eliminate the unknown terms (i.e., propagation delay $U + P_{d_1} + P_{d_2}$):

$$\begin{aligned} rtt_i - rtt_{\min} &= q_i + t_i - \min\{q_i + t_i | \forall i\} \\ &< q_i + t_i \qquad \because q_i \geq 0, t_i > 0. \end{aligned} \tag{23}$$

Rearranging terms we then have a lower bound for $q_i$:

$$q_i > rtt_i - rtt_{\min} - t_i \tag{24}$$

with all three terms in the R.H.S. known to the sender.

To relate (24) to $n_i$ we note that $q_i$ must also satisfy the upper bound in (20). Hence we can obtain an estimate of $n_i$ by increasing it from $n_i = 0, 1, \ldots,$ until the R.H.S. of (20) becomes larger than the lower bound in (24), i.e., satisfying both bounds:

$$n_i' = \min\left\{ n_i \left| \sum_{k=i-n_i-1}^{i-1} t_k \geq (rtt_i - rtt_{\min} - t_i), \forall n_i = 0, 1, \ldots \right. \right\}. \tag{25}$$

The smallest $n_i$ that satisfies both bounds then become the estimated queue length $n_i'$.

Note that in practice TCP receiver may implement delayed ACK, generating only one ACK for every two packets received. To tackle this we can compute (25) only for cases where the required $t_i$ (and hence $r_{i-1}$) is available, i.e., roughly at half the frequency. The summation term in (25) may not be affected though as we can compute the sum of say, $t_{i-1}$ and $t_{i-2}$ from $r_{i-1} - r_{i-3}$, without the need for $r_{i-2}$. In the worst case the estimation will be off by at most one packet.

Finally, the link buffer size can then be computed from the maximum of all queue length estimates:

$$L' = \max\left\{ n_i' | \forall i \right\} + 1. \tag{26}$$

We thoroughly evaluate SoD's estimation accuracy in the next section.

## 4 Performance Evaluation

In this section we evaluate and compare the estimation accuracy of SoD to existing algorithms using a simulator implemented using NS2 [14]. It is worth noting that while NS2 already has built-in TCP implementations, their implementations do deviate from the actual TCP implementations used in the Internet today in important ways. Therefore we chose to implement the simulator based on actual TCP implementations in current Linux kernels and also implemented widely-deployed optional TCP features such as TCP SACK [15] to establish a more realistic platform for the simulations.

The simulated network topology is depicted in Fig. 7 and Table 2 summarizes the simulation parameters. For passive estimation, we employed TCP as the transport and the sender simply sent data as fast as TCP allowed. We incorporated the TCP CUBIC [16] kernel module from Linux 2.6 into the simulator as it is one of the most widely used TCP variant in use in the Internet. Additionally, we implemented TCP SACK [15] in the simulator to more accurately model real TCP behavior. For comparison purposes we also implemented SoD over TCP Vegas [7] by incorporating its Linux kernel module into NS2.

Finally, as the open source Linux kernel does not currently implement FAST TCP we employed the FAST TCP NS2 module developed by the CUBIN Lab [17] in our simulations.

TABLE 3
Comparison of Relative Link Buffer Size Estimation Errors for (a) Active, and (b) Passive Estimation with Fixed Bottleneck Link Bandwidth

| $L$ | Max-Min (%) | | Loss-Pair (%) | | Sum-of-Delays (%) | |
|---|---|---|---|---|---|---|
| | $E$ | stdev | $E$ | stdev | $E$ | stdev |
| 100 | -1.00 | 0.00 | -1.26 | 0.15 | 0.00 | 0.00 |
| 200 | -0.50 | 0.01 | -0.63 | 0.08 | 0.00 | 0.00 |
| 300 | -0.33 | 0.00 | -0.41 | 0.04 | 0.00 | 0.00 |
| 400 | -0.25 | 0.00 | -0.30 | 0.04 | 0.00 | 0.00 |
| 500 | -0.20 | 0.00 | -0.24 | 0.03 | 0.00 | 0.00 |
| 600 | -0.16 | 0.00 | -0.21 | 0.03 | 0.00 | 0.00 |
| 700 | -0.14 | 0.00 | -0.17 | 0.02 | 0.00 | 0.00 |
| 800 | -0.12 | 0.00 | -0.15 | 0.02 | 0.00 | 0.00 |
| 900 | -0.11 | 0.00 | -0.13 | 0.02 | 0.00 | 0.00 |
| 1000 | -0.10 | 0.00 | -0.12 | 0.02 | 0.00 | 0.00 |

(a)

| $L$ | Max-Min (%) | | Loss-Pair (%) | | Sum-of-Delays (%) | |
|---|---|---|---|---|---|---|
| | $E$ | stdev | $E$ | stdev | $E$ | stdev |
| 100 | 0.72 | 1.03 | -1.30 | 0.35 | 0.33 | 0.50 |
| 200 | -0.07 | 0.47 | -1.00 | 0.27 | 0.56 | 0.53 |
| 300 | -0.32 | 0.35 | -0.94 | 0.25 | 0.22 | 0.33 |
| 400 | -0.41 | 0.22 | -0.89 | 0.24 | 0.14 | 0.18 |
| 500 | -0.47 | 0.15 | -0.86 | 0.22 | 0.09 | 0.15 |
| 600 | -0.51 | 0.16 | -0.82 | 0.22 | 0.15 | 0.15 |
| 700 | -0.51 | 0.10 | -0.77 | 0.20 | 0.08 | 0.14 |
| 800 | -0.48 | 0.05 | -0.69 | 0.16 | 0.07 | 0.17 |
| 900 | -0.58 | 0.15 | -0.79 | 0.23 | 0.07 | 0.11 |
| 1000 | -0.62 | 0.18 | -0.75 | 0.22 | 0.03 | 0.05 |

(b)

### 4.1 Link Buffer Size Estimation for Fixed-Bandwidth Link

We first consider the baseline case of estimating link buffer size in a network with fixed bottleneck link bandwidth using active estimation. Table 3(a) and Fig. 8(a) compare the estimation error of max-min, loss-pair, and sum-of-delays algorithms. Not surprisingly both max-min and loss-pair performed well in this setup, with estimation errors below 2% in all cases. The proposed sum-of-delays method achieved very accurate estimation with zero error (to two decimal places). Moreover, its accuracy is consistent across

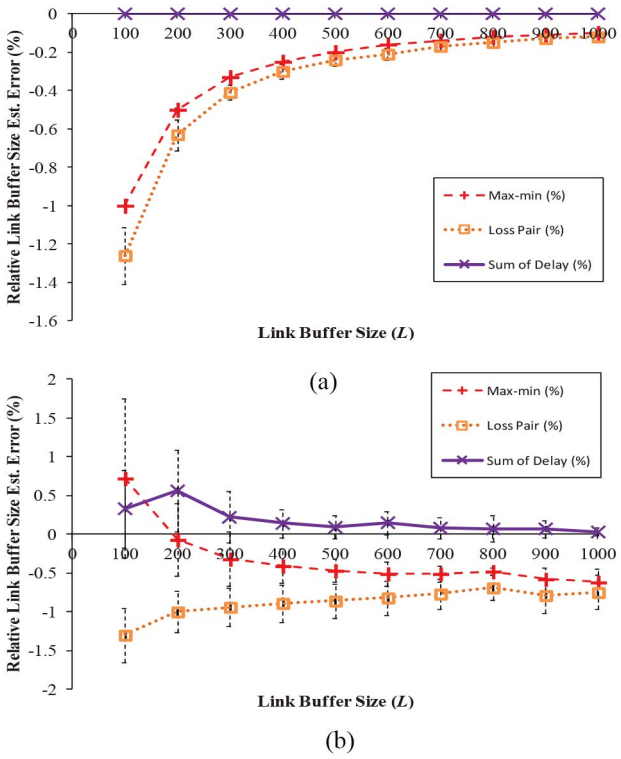



Fig. 8. Comparison of relative link buffer size estimation errors for (a) active estimation, and (b) passive estimation with fixed bottleneck link bandwidth (link buffer size, relative error, fixed bandwidth).

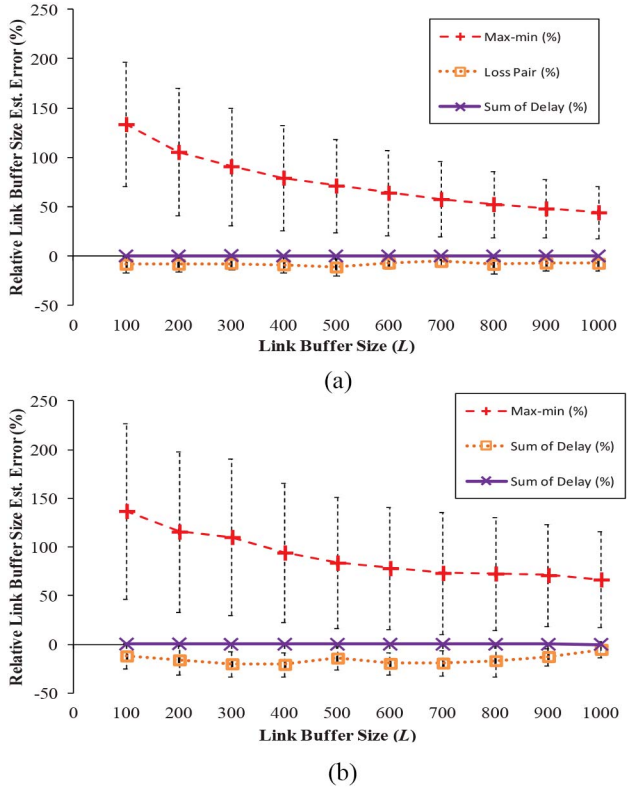



Fig. 9. Comparison of relative link buffer size estimation errors for (a) active estimation, and (b) passive estimation with varying bottleneck link bandwidth with error bars depicting the standard deviation (link buffer size, relative error, variable bandwidth).

all link buffer sizes, as opposed to max-min/loss-pair where the estimation error increases with smaller link buffer sizes.

Next we investigate passive estimation where no explicit measurement packets are generated. The estimation is computed solely from the timings of normal data packets which are controlled and scheduled according to TCP CUBIC's flow and congestion control algorithms. While this is not the original intended applications for max-min/loss pair, the results in Table 3(b) and Fig. 8(b) show that their performances were not significantly worse than their active estimation counterparts. The proposed SoD algorithm still outperformed max-min/loss-pair for most of the link buffer size ranges (e.g., from 300 to 1000).

### 4.2 Link Buffer Size Estimation for Variable-Bandwidth Link

For bandwidth-varying network simulation runs in this and the following sections, we used 10 sets of actual bandwidth trace data captured from a production 3G/HSPA network to modulate the bandwidth of the bottleneck link in the simulator (see Fig. 8). The trace data were captured at a mobile computer connected to a production 3G/HSPA network using a USB 3G/HSPA modem. Thus the captured traffic traces are subject to and representative of real-world network conditions such as cross traffic generated by other mobile users sharing the same cell, radio signal quality fluctuations, interference from other devices, etc. The client computer remained stationary during the trace capture process, thus it did not yet incorporate the effect of mobility-induced bandwidth fluctuations, which is a subject for future research.

Table 4(a) and (b) compare the mean and standard deviation of the estimation errors of the three estimation algorithms under active and passive estimations respectively. The results are also presented visually in Fig. 9(a) and (b), with the standard deviation indicated by the error bars.

First, compared to the fixed-bandwidth case the performance of max-min degraded significantly under the varying-bandwidth case. This is especially significant at smaller link buffer sizes (e.g., $L = 100, 200$) where the estimation errors exceeded 100%. This is due to bandwidth variation which resulted in significant variations in the transmission delays. As the max-min method is based on the difference between maximum and minimum RTTs the transmission delay variations significantly distorted the link buffer size estimates.

Second, the loss-pair algorithm performed substantially better than max-min under active estimation. This is because loss-pair does not base its estimation on the worst-case measurement, but determines the mathematical mode of the measured samples to filter out noises [6].

Comparing the passive estimation results in Table 4(b) to the active estimation results in Table 4(a), we observe that the estimation errors increase further for both max-min and loss-pair algorithms. By contrast, SoD achieved a low level of estimation errors, smaller than 0.8 in all cases, which are two orders of magnitudes lower than

TABLE 4
Comparison of Relative Link Buffer Size Estimation Errors for (a) Active, and (b) Passive Estimation with Varying Bottleneck Link Bandwidth

| $L$ | Max-Min (%) | | Loss-Pair (%) | | Sum-of-Delays (%) | |
|---|---|---|---|---|---|---|
| | $E$ | stdev | $E$ | stdev | $E$ | stdev |
| 100 | 133.70 | 63.03 | -8.39 | 8.29 | 0.00 | 0.00 |
| 200 | 105.54 | 64.66 | -8.38 | 7.43 | 0.00 | 0.00 |
| 300 | 90.94 | 59.46 | -7.66 | 6.47 | 0.00 | 0.00 |
| 400 | 79.28 | 53.64 | -9.13 | 8.02 | 0.00 | 0.00 |
| 500 | 71.33 | 47.68 | -11.21 | 9.10 | 0.00 | 0.00 |
| 600 | 64.15 | 42.81 | -7.25 | 5.17 | 0.00 | 0.00 |
| 700 | 57.60 | 38.24 | -5.58 | 4.93 | 0.00 | 0.00 |
| 800 | 52.47 | 33.56 | -8.59 | 8.74 | 0.00 | 0.00 |
| 900 | 48.12 | 29.77 | -7.31 | 7.54 | 0.00 | 0.00 |
| 1000 | 44.20 | 26.68 | -7.51 | 7.51 | 0.00 | 0.00 |

(a)

| $L$ | Max-Min (%) | | Loss-Pair (%) | | Sum-of-Delays (%) | |
|---|---|---|---|---|---|---|
| | $E$ | stdev | $E$ | stdev | $E$ | stdev |
| 100 | 136.93 | 90.43 | -11.96 | 12.57 | 0.80 | 0.42 |
| 200 | 115.74 | 82.12 | -15.96 | 14.95 | 0.35 | 0.24 |
| 300 | 110.24 | 80.55 | -20.01 | 12.72 | 0.37 | 0.33 |
| 400 | 94.25 | 71.36 | -20.6 | 12.38 | 0.23 | 0.28 |
| 500 | 83.86 | 67.03 | -14.11 | 11.62 | 0.16 | 0.18 |
| 600 | 78.31 | 63.09 | -19.57 | 11.46 | 0.12 | 0.11 |
| 700 | 73.34 | 62.63 | -19.26 | 13.25 | 0.07 | 0.14 |
| 800 | 72.75 | 57.79 | -16.76 | 16.74 | 0.11 | 0.11 |
| 900 | 71.10 | 51.99 | -12.62 | 8.79 | 0.09 | 0.09 |
| 1000 | 66.66 | 49.00 | -5.17 | 8.44 | 0.09 | 0.09 |

(b)

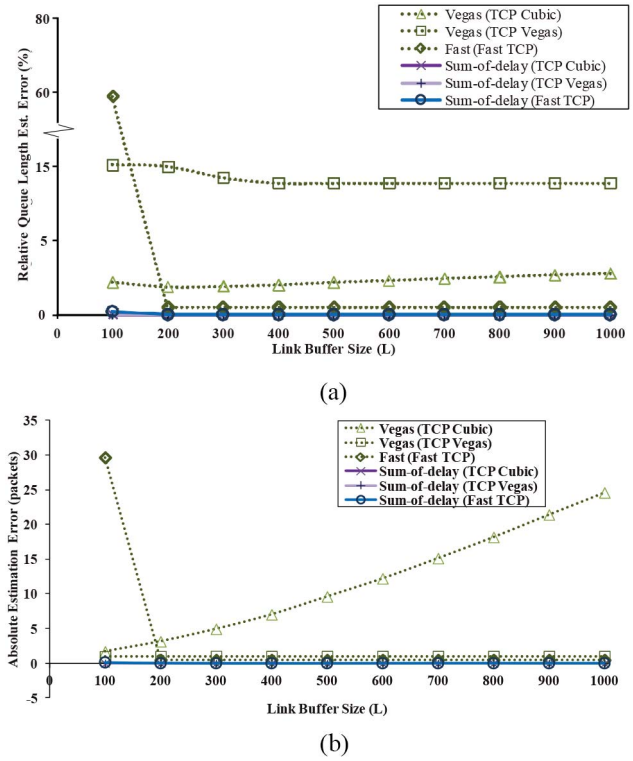



Fig. 10. Comparison of (a) relative, and (b) absolute queue length estimation errors in different TCP variants with fixed bottleneck link bandwidth (queue length, relative error, fixed bandwidth).

max-min and loss-pair. The reason for SoD's significant improvements in estimation accuracy is due to its use of the *measured* transmission times which inherently incorporated the effect of bandwidth variations in computing the link buffer size. In contrast, max-min and loss-pair estimated the link buffer size from a constant link capacity (see the constant $C$ in (10)) which introduces substantial errors when the underlying link bandwidth is no longer constant.

### 4.3 Queue Length Estimation for Fixed-Bandwidth Link

We evaluate queue length estimation algorithms over TCP flows, i.e., the data flow is regulated by the underlying TCP's flow and congestion control modules. Between three queue-length estimation algorithms (Vegas, FAST, and SoD) and three TCP variants (CUBIC, Vegas, and FAST), we investigated six combinations:

- *Vegas over TCP CUBIC* – queue length estimated using the Vegas method during the congestion-avoidance phase of a TCP CUBIC flow;
- *Vegas over TCP Vegas* – queue length estimated using the Vegas method during the congestion-avoidance phase of a TCP Vegas flow;
- *FAST over FAST TCP* – queue length estimated using the FAST method during the slow-start and congestion-avoidance phases of a FAST TCP flow;
- *SoD over TCP CUBIC* – queue length estimated using the SoD method during all three phases, i.e., slow-start, congestion-avoidance, and loss-recovery phases of a TCP CUBIC flow;
- *SoD over TCP Vegas* – queue length estimated using the SoD method during all three phases, i.e., slow-start, congestion-avoidance, and loss-recovery phases of a TCP Vegas flow;
- *SoD over FAST TCP* – queue length estimated using the SoD method during all three phases, i.e., slow-start, congestion-avoidance, and loss-recovery phases of a FAST TCP flow.

Fig. 10(a) and (b) compare the relative and absolute estimation errors for the six combinations in fixed bottleneck link bandwidth networks. Except for the case with link buffer size of 100 packets, the FAST algorithm outperforms the two cases with the Vegas estimation algorithm. The FAST algorithm's anomaly is again due to its target queue length setting, 100 packets by default, too close to the link buffer size, thus causing frequent buffer overflows.

By contrast, the proposed SoD method worked well over all TCP variants. Table 5 shows that the estimation errors are no more than 0.1% for all three TCP variants over all link buffer sizes, demonstrating SoD's consistent performance despite substantial variations in the underlying transport protocol.

### 4.4 Queue Length Estimation for Variable-Bandwidth Link

Next we investigate the algorithms' performance in mobile networks using trace data captured from production 3G/HSPA networks as described in Section 4.2.

Fig. 11(a) and (b) plot the relative and absolute estimation errors for the six combinations over mobile data

TABLE 5
Comparison of Relative Queue Length Estimation Errors for Different TCP Variants Under Fixed Bottleneck Link Bandwidth

| *Type of TCP* | TCP CUBIC | | TCP Vegas | | FAST TCP | |
|---|---|---|---|---|---|---|
| $L$ | Vegas (%) | SOD (%) | Vegas (%) | SOD (%) | FAST (%) | SOD (%) |
| 100 | 2.19 | 0.00 | 15.65 | 0.00 | 48.03 | 0.20 |
| 200 | 1.87 | 0.00 | 14.94 | 0.00 | 0.48 | 0.00 |
| 300 | 1.91 | 0.00 | 13.45 | 0.00 | 0.48 | 0.00 |
| 400 | 2.01 | 0.00 | 12.72 | 0.00 | 0.48 | 0.00 |
| 500 | 2.18 | 0.00 | 12.72 | 0.00 | 0.48 | 0.00 |
| 600 | 2.30 | 0.00 | 12.72 | 0.00 | 0.48 | 0.00 |
| 700 | 2.43 | 0.00 | 12.72 | 0.00 | 0.48 | 0.00 |
| 800 | 2.54 | 0.00 | 12.72 | 0.00 | 0.48 | 0.00 |
| 900 | 2.67 | 0.00 | 12.72 | 0.00 | 0.48 | 0.00 |
| 1000 | 2.75 | 0.00 | 12.72 | 0.00 | 0.48 | 0.00 |

networks. The same results are also listed in Table 6. Compared to the fixed bandwidth cases in Section 3, the estimation errors for all three existing estimation algorithms increased significantly in variable bandwidth networks. The degradation in estimation accuracy for FAST over FAST TCP is particularly significant (e.g., from 0.48% to 14.3% for $L = 300$). The reason for this is not yet clear and further analysis of FAST TCP's algorithm is warranted to identify the causes.

In comparison, all SoD cases achieved consistently low level of estimation errors, under 1% in all cases, thus outperforming existing algorithms by multiple orders of magnitude. The reason for SoD's higher estimation accuracy is again due to its use of past *measured* transmission times in computing the queue length estimates. By contrast, both Vegas and FAST algorithms rely on the congestion window size as an indirect estimate of the link bandwidth in estimating the queue length (see (12)). This is inherently inaccurate as the congestion window often underestimates the link bandwidth during TCP's slow-start phase, and it will also eventually overestimate the link bandwidth near the end of the congestion-avoidance phase. Hence such errors in tracking the link bandwidth, especially in the presence of bandwidth variations, substantially degraded the accuracy in estimating the queue length.

TABLE 6
Comparison of Relative Queue Length Estimation Errors for Different TCP Variants Under Varying Bottleneck Link Bandwidth

| *Type of TCP* | TCP CUBIC | | TCP Vegas | | FAST TCP | |
|---|---|---|---|---|---|---|
| $L$ | Vegas (%) | SOD (%) | Vegas (%) | SOD (%) | FAST (%) | SOD (%) |
| 100 | 6.69 | 0.08 | 17.16 | 0.74 | 70.32 | 1.49 |
| 200 | 4.21 | 0.03 | 17.12 | 0.72 | 14.40 | 0.20 |
| 300 | 3.62 | 0.02 | 17.27 | 0.72 | 14.39 | 0.20 |
| 400 | 3.51 | 0.02 | 17.63 | 0.71 | 14.39 | 0.20 |
| 500 | 3.49 | 0.01 | 15.80 | 0.60 | 14.39 | 0.20 |
| 600 | 3.62 | 0.01 | 15.80 | 0.60 | 14.39 | 0.20 |
| 700 | 3.69 | 0.01 | 15.80 | 0.60 | 14.39 | 0.20 |
| 800 | 3.80 | 0.01 | 15.80 | 0.60 | 14.39 | 0.20 |
| 900 | 3.88 | 0.01 | 15.80 | 0.60 | 14.39 | 0.20 |
| 1000 | 3.95 | 0.01 | 15.80 | 0.60 | 14.39 | 0.20 |

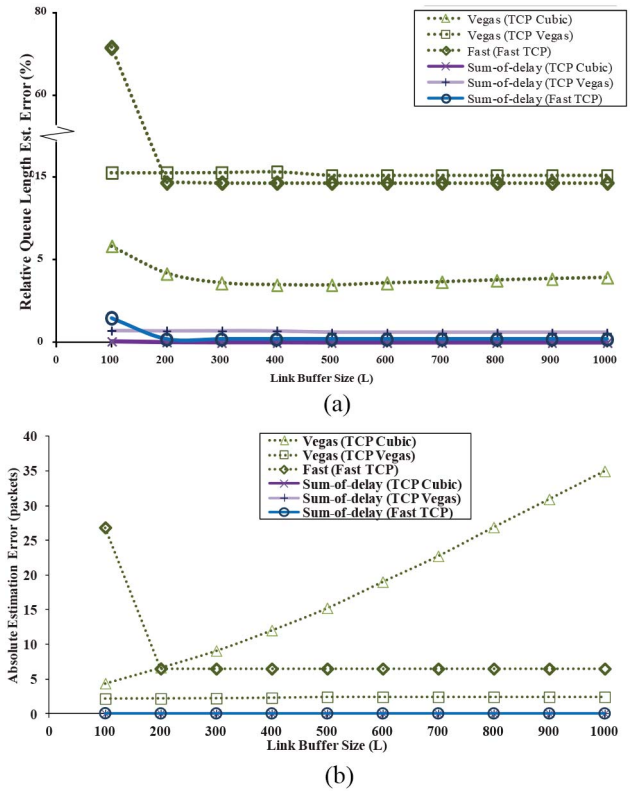



Fig. 11. Comparison of (a) relative, and (b) absolute queue length estimation errors in different TCP variants with varying bottleneck link bandwidth (queue length, relative error, variable bandwidth).

All of the passive estimations in Section 4.1 to 4.4 are based on TCP data packet timings. However TCP flows often cannot fully utilize the available link bandwidth, typically achieving only 30% to 80% link utilization depending on the link properties. This implies that the bottleneck link must be idle for a substantial amount of time, thereby violating the assumption of zero link idle time in the SoD algorithm (see (17)). Nevertheless the results in these sections clearly show that its impact is negligible given SoD's consistent performance over the three TCP variants tested.

### 4.5 Impact of Link Delay Variations

So far the simulations were performed over the topology in Fig. 7 with constant link propagation delays. In practice, mobile networks do exhibit some variations in the network delay due to link-layer retransmissions, channel contention, or uplink bandwidth fluctuations. This will affect the measured ACK timings and depending on the extent of delay variations we expect it to degrade the accuracy of link buffer size and queue length estimations.

Our measurements of production 3G networks show that the actual uplink delay variations vary depending on network conditions, network types, locations, etc., with standard deviations ranging from 10 to 15 ms. As an exact model for the uplink delay variation is not available we approximate the delay variations by replacing the constant delay in the uplink (i.e., $U$) with a random delay variation which is normally-distributed with standard deviations ranging from 0 to 20 ms. We also conducted separate

TABLE 7
Comparison of Relative Link Buffer Size Estimation Errors for Passive Estimation with Varying Bottleneck Link Bandwidth and Varying Uplink Delay (SoD-TS Implements Uplink Delay Variation Compensation Using TCP Timestamps)

| $\sigma$ | Max-Min (%) | | Loss-Pair (%) | | SoD (%) | | SoD-TS (%) | |
|---|---|---|---|---|---|---|---|---|
| | $E$ | stdev | $E$ | stdev | $E$ | stdev | $E$ | stdev |
| 0 | 66.66 | 49.00 | -5.17 | 8.44 | 0.09 | 0.09 | 0.20 | 0.00 |
| 5 | 61.56 | 48.88 | -7.51 | 10.41 | 0.62 | 0.43 | 0.20 | 0.00 |
| 10 | 60.01 | 45.78 | -10.01 | 11.43 | 1.77 | 0.58 | 0.20 | 0.00 |
| 15 | 59.44 | 42.52 | -9.54 | 10.57 | 2.76 | 1.06 | 0.22 | 0.04 |
| 20 | 55.66 | 38.14 | -8.56 | 9.26 | 3.62 | 1.19 | 0.98 | 0.65 |

simulations using actual uplink delay trace data and found the results to be consistent, with SoD performing slightly better under the trace-driven simulations.

Table 7 summarizes the results for link buffer size estimation. The results show that delay variations do have some impact to the estimation accuracy. Nonetheless in all cases simulated, SoD not only achieved lower estimation error than min-max and loss-pair, but also exhibited significantly lower variations in estimation errors.

Table 8 summarizes the results for queue length estimation. Under TCP CUBIC both the Vegas and SoD estimation algorithms degraded slightly with higher delay variations. At the range of delay variations in practice (i.e., $\sigma - 10 \sim 15$ ms) SoD substantially outperformed the Vegas algorithm and is able to maintain the estimation error to below 2%. By contrast, the cases for TCP Vegas and FAST TCP all degraded substantially in the presence of delay variations, with TCP Vegas particularly significant. This is due to TCP Vegas' design to maintain only a small number of packets in the bottleneck link. Consequently the queueing delay under TCP Vegas is much shorter than other TCP variants, and thus the errors introduced by delay variations are relatively more significant.

Overall, in the presence of delay variations, SoD can still achieve significantly better accuracy for link buffer size estimation (1~2 order of magnitude). For queue length estimation, SoD's accuracy is robust when applied to TCP CUBIC, which is one of the most widely deployed TCP variants in the Internet. The applications to TCP Vegas and FAST, however, are less positive.

There is a solution to the problem of uplink delay variations – TCP's timestamp option [18]. This option, which is widely implemented by mobile devices [19], includes two timestamp fields in the TCP header for storing the transmission timestamp and for echoing the timestamp received from the peer host. Using the receiver's ACK timestamps and the ACK arrival times, the sender can then estimate the uplink delay variations and compensate for them in computing the link buffer size and queue length. We implemented this into the simulator (including receiver timestamp clock rate estimation) and summarized the results under SoD-TS in Table 7 and 8. Using TCP timestamp SoD achieves significantly lower estimation errors for both link buffer size and queue length estimation in the presence of uplink delay variations. The only requirements are the availability of TCP timestamp implementation [19] and sufficiently fine timestamp granularity.

TABLE 8
Comparison of Relative Queue Length Estimation Errors for Different TCP Variants Under Varying Bottleneck Link Bandwidth and Varying Uplink Delay (SoD-TS Implements Uplink Delay Variation Compensation Using TCP Timestamps)

| $\sigma$ | TCP CUBIC | | | TCP Vegas | | FAST TCP | |
|---|---|---|---|---|---|---|---|
| | Vegas (%) | SoD (%) | SoD-TS (%) | Vegas (%) | SoD (%) | FAST (%) | SoD (%) |
| 0 | 3.95 | 0.01 | 0.042 | 15.80 | 0.60 | 6.69 | 0.06 |
| 5 | 4.03 | 0.44 | 0.042 | 104.61 | 119.82 | 8.70 | 5.10 |
| 10 | 4.23 | 1.03 | 0.043 | 140.76 | 159.28 | 11.25 | 10.46 |
| 15 | 4.78 | 1.88 | 0.043 | 144.72 | 162.53 | 15.95 | 19.28 |
| 20 | 4.97 | 2.43 | 0.046 | 140.68 | 157.74 | 17.34 | 24.49 |

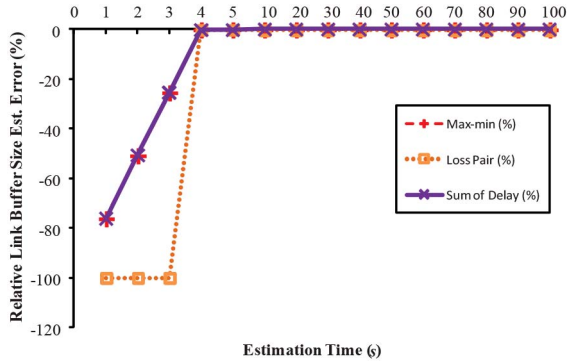



Fig. 12. Comparison of estimation errors versus estimation duration for active estimation with fixed bottleneck link bandwidth of 7.2Mbps.

### 4.6 Convergence

It takes time for the estimation algorithms to collect measurement data to improve estimation accuracy. In this section we investigate the convergent rates for max-min, loss-pair, and SoD in both fixed and variable bandwidth networks, using active and passive estimation.

We first consider the baseline scenario of active estimation in a fixed-bandwidth network. Fig. 12 plots the relative estimation error for bottleneck link bandwidth of 7.2Mbps. There are two observations. First, all three algorithms remained accurate once they converged. Second, max-min and SoD converged faster than loss-pair, due to the latter's design to determine the estimation from the mathematical mode of the collected samples.

Next we consider the case of passive estimation (over TCP CUBIC) in a fixed-bandwidth network. Fig. 13 plots the relative estimation error for bottleneck link bandwidth of 7.2Mbps. Compare to the active estimation case SoD converged faster than max-min as the latter depends on accurate estimate of the link bandwidth. In passive estimation TCP begins with a low transmission rate (TCP Slow-Start) and hence the link bandwidth parameter cannot be accurately estimated until TCP has ramped up its transmission rate at a later time.

Moving on to varying bandwidth environment the results are more interesting. Figs. 14 and 15 compare the algorithms in active and passive estimation respectively. One notable observation is that the max-min algorithm did not converge at all. Beginning with an underestimated link

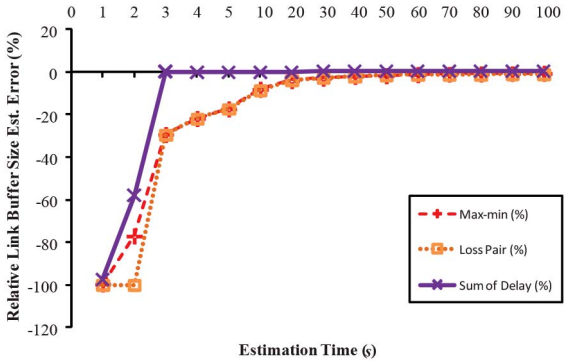



Fig. 13. Comparison of estimation errors versus estimation duration for passive estimation with fixed bottleneck link bandwidth of 7.2Mbps.

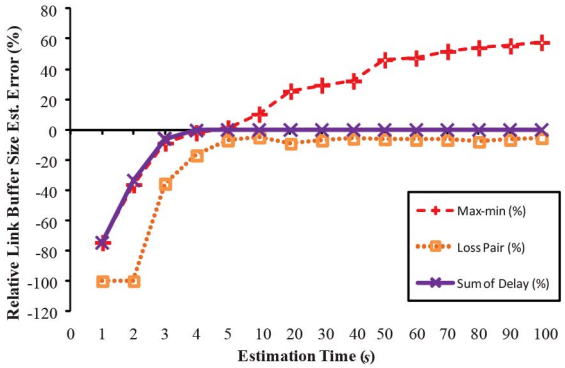



Fig. 14. Comparison of estimation errors versus estimation duration for active estimation with varying bottleneck link bandwidth.

buffer size, the estimated link buffer size increased continuously to become overestimation after about 5 seconds. The reason being that max-min assumes constant packet transmission time which is not true in a bandwidth-varying network. The bandwidth variations modulated the transmission time, thus introduced errors into the estimated result. As max-min's estimation is based on the difference between maximum and minimum measurement samples the difference will clearly increase with more samples collected. This again shows that the max-min algorithm, not being designed for bandwidth-varying networks, is not suitable for use in mobile data networks. By contrast, the proposed SoD algorithm performed consistently and was able to arrive at an accurate estimation within 4~5 seconds, even in the extreme case of passive estimation over bandwidth-varying network.

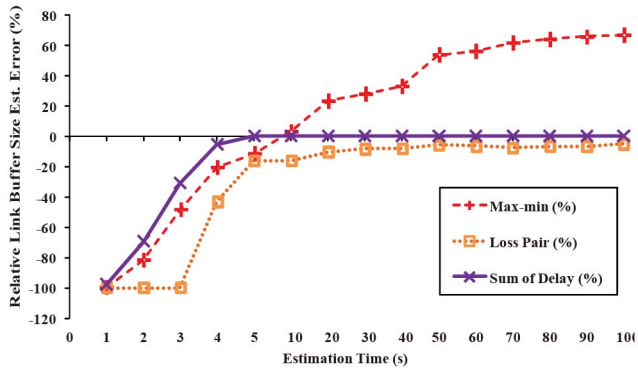



Fig. 15. Comparison of estimation errors versus estimation duration for passive estimation with varying bottleneck link bandwidth.

### 4.7 Limitations in Passive Estimation

As discussed in Section 2.2 and 2.3 the Vegas queue-length estimation algorithm is limited to TCP's congestion-avoidance phase, and FAST TCP's algorithm limited to TCP's slow-start and congestion-avoidance phases. By contrast, the proposed SoD algorithm can be applied to all three phases of TCP as it does not rely on TCP's congestion window in its estimation.

To further investigate their performances in different TCP phases we table the estimation errors separately for the three TCP phases in Tables 9 and 10 for fixed ($C = 7.2$Mbps, $L = 700$ packets) and variable (averaged over 10 sets of bandwidth traces) bandwidth cases respectively. Not only that SoD works for all three phases, it consistently outperformed Vegas and FAST algorithms by multiple orders of magnitude. SoD's consistent accuracy in the loss recovery phase will be a particularly desirable property in mobile data networks in cases where random packet losses cannot be fully recovered by link-layer retransmissions. Devising ways to exploit this property for use in mobile data networks warrants further research.

Finally, as shown from the above simulation results, the proposed sum-of-delays algorithm consistently achieved very low estimation errors in both link buffer size and queue length estimation, no more than 2% in most cases, even under the most challenging conditions. This confirms that by incorporating the measured transmission delays in estimating the link buffer size and queue length, we can effectively incorporate and compensate for bandwidth variations at the bottleneck link.

TABLE 9
Relative Estimation Errors for Queue Length Estimation in Different TCP Variants Under Fixed Bottleneck Link Bandwidth

| Algorithms | TCP Phases | | |
|---|---|---|---|
| | Slow-Start | Congestion Avoidance | Loss Recovery |
| Vegas (TCP CUBIC) | N/A | 2.43 | N/A |
| Vegas (TCP Vegas) | N/A | 12.72 | N/A |
| FAST (FAST TCP) | 20.62 | 0.48 | N/A |
| SoD (TCP CUBIC) | 0.03 | 0.01 | 0.01 |
| SoD (TCP Vegas) | 0.87 | 0.01 | N/A (no loss) |
| SoD (Fast TCP) | 0.02 | 0.01 | N/A (no loss) |

TABLE 10
Relative Estimation Errors for Queue Length Estimation in Different TCP Variants Under Varying Bottleneck Link Bandwidth

| Algorithms | TCP Phases | | |
|---|---|---|---|
| | Slow-Start | Congestion Avoidance | Loss Recovery |
| Vegas (TCP CUBIC) | N/A | 3.69 | N/A |
| Vegas (TCP Vegas) | N/A | 15.80 | N/A |
| FAST (FAST TCP) | 19.02 | 6.69 | N/A |
| SoD (TCP CUBIC) | 0.01 | 0.01 | 0.02 |
| SoD (TCP Vegas) | 0.55 | 0.60 | N/A (no loss) |
| SoD (Fast TCP) | 0.10 | 0.06 | N/A (no loss) |

## 5 Summary and Future Work

This work developed a novel sum-of-delays algorithm for estimating link buffer size and queue length in bandwidth-varying networks such as mobile data networks. Knowledge of the link buffer size can be exploited at various layers to enhance the performance of network applications, or for monitoring/management of the network. For example, transport layer protocols such as TCP can be modified to integrate the sum-of-delays algorithm to perform passive estimation of the link buffer size so that the latter can be exploited in its congestion control algorithm to improve bandwidth utilization while still preventing network congestion. Our initial results showed that by utilizing the estimated queue length in congestion control (as opposed to using packet loss events) we can potentially improve the TCP throughput by over 50% compared to TCP CUBIC even at a random packet loss rate as low as 0.1%. A full treatment of the topic is beyond the scope of this paper and will be reported in a separate work.

At the application layer, the sum-of-delays algorithm can also be employed to incorporate the link buffer size information into congestion control, resource allocation, traffic policing, rate adaptation, error control, and so on. Much work remains to be done to fully explore the potentials of such applications.

The proposed sum-of-delays method can be extended in two directions. First, the version investigated in this work relies on the sender to carry out the measurements and link buffer size estimation – sender-based estimation. In certain applications it may not be practical to modify the sender and in such cases it will require the estimation to be done at the receiver – receiver-based estimation. Whether receiver-based estimation is feasible and its performance compared to sender-based estimation is a subject for future research.

Second, the sum-of-delays method could also be used to continuously estimate the queue length at the bottleneck link in addition to estimating the link buffer size. Knowledge of the queue length will be useful to the design of new congestion control algorithm for use in both transport and application protocols. In addition to bandwidth probing and congestion recovery, it is also possible to employ queue length information to further improve the packet loss recovery mechanism, to implement non-uniform bandwidth sharing mechanisms, etc.

Finally, unlike packet loss, queue-length information is an early indicator of network conditions, and as such, could be very useful to application-layer adaptation mechanisms such as adaptive video stream, multimedia conferencing, and any applications which are bandwidth- and delay-sensitive. The proposed Sum-of-Delay algorithm offers a new avenue for implementing intelligent application-layer adaptation algorithms that can proactively react to changes in network conditions (as opposed to passively react to) to provide consistent quality-of-service over current and future mobile data networks.

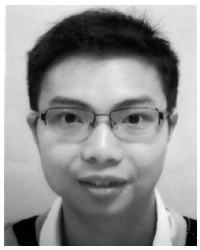
## Acknowledgments

The authors sincerely thank the associate editor and the anonymous reviewers for their insightful comments and suggestions in improving this paper. The experimental results in this paper were made possible through the support of our industry collaborators. They sincerely thank them for their generous support. This work was funded in part by the Innovation and Technology Fund (ITS/014/10 and ITS/146/11) provided by the Innovation and Technology Commission, HKSAR[1]. An early version of this work covering only link buffer size estimation was published in the 7$^{th}$ IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2011), Shanghai, China, October 10–12, 2011.

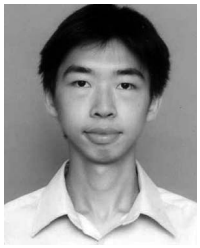
## References

[1] A. Akella, S. Seshan, and A. Shaikh, "An empirical evaluation of wide-area internet bottlenecks," in *Proc. ACM IMC*, New York, NY, USA, Oct. 2003.
[2] K. Lakshminarayanan and V. Padmanabhan, "Some findings on the network performance of broadband hosts," in *Proc. ACM IMC*, New York, NY, USA, Oct. 2003.
[3] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell, "pathChirp: Efficient available bandwidth estimation for network paths," in *Proc. PAM*, 2003.
[4] M. Hirabaru, "Impact of bottleneck queue size on TCP protocols and its measurement," *IEICE Trans. Inf. Syst.*, vol. E89-D, no. 1, Jan. 2006, pp. 132–138.
[5] M. Claypool, R. Kinicki, M. Li, J. Nichols, and H. Wu, "Inferring queue sizes in access networks by active measurement," in *Proc. 5th PAM*, Antibes Juan-les-Pins, France, 2004.
[6] J. Liu and M. Crovella, "Using loss pairs to discover network properties," in *Proc. ACM SIGCOMM*, New York, NY, USA, 2001, pp. 127–138.
[7] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP vegas: New techniques for congestion detection and avoidance," in *Proc. SIGCOMM*, London, U.K., Oct. 1994, pp. 24–35.
[8] S. Hegde *et al.*, "Fast TCP in high speed networks: An experimental study," in *Proc. GridNets*, San Jose, CA, USA, Oct. 2004.
[9] C. P. Fu and S. C. Liew, "TCP veno: TCP enhancement for wireless access networks," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 2, pp. 216–228, Feb. 2003.
[10] V. K. Garg, *Wireless Network Evolution: 2G to 3G*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.
[11] E. Dahiman, S. Parkvall, J. Skold, and P. Beming, *3G Evolution: HSPA and LTE for Mobile Broadband*, 2nd ed. Boston, MA, USA: Academic Press, 2008.
[12] D. Astely *et al.*, "LTE: The evolution of mobile broadband," *IEEE Commun. Mag.*, vol. 47, no. 4, pp. 44–51, Apr. 2009.
[13] K. Liu and J. Y. B. Lee, "Mobile accelerator: A new approach to improve TCP performance in mobile data networks," in *Proc. 7th IEEE IWCMC*, Istanbul, Turkey, Jul. 2011.
[14] NS2 *Network Simulator* [Online]. Available: http://www.isi.edu/nsnam/ns/
[15] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, "An extension to the selective acknowledgement (SACK) option for TCP," *RFC 2883*, 2000.
[16] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," in *Proc. Int. Workshop Protocols FAST Long Distance Netw.*, New York, NY, USA, 2005.
[17] *FAST TCP ns2 Module* [Online]. Available: http://www.cubinlab.ee.unimelb.edu.au/ns2FASTtcp/
[18] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," *RFC 1323*, May 1992.
[19] E. Halepovic, J. Pang, and O. Spatscheck, "Can you GET me now? Estimating the time-to-first-byte of HTTP transactions with passive measurements," in *Proc. ACM Conf. IMC*, Boston, MA, USA, Nov. 2012, pp. 115–122.

1. Any opinions, findings, conclusions or recommendations expressed in this material/event (or by members of the project team) do not reflect the views of the Government of the Hong Kong Special Administrative Region, the Innovation and Technology Commission or the assessment panel of the Innovation and Technology Support Programme of the Innovation and Technology Fund.



**Stanley C. F. Chan** received his B.Eng. and M.Phil. degrees in information engineering from the Chinese University of Hong Kong, Shatin, Hong Kong, in 2008 and 2011, respectively. From 2008 to 2011, he was a member of the Multimedia Communications Laboratory of CUHK where he participated in the research of mobile communications and protocol optimization. He currently works in the industry in Hong Kong.



**K. M. Chan** received his B.Eng. degree in information engineering from the Chinese University of Hong Kong, Shatin, Hong Kong, in 2010. He is currently a research student at the Multimedia Communications Laboratory of CUHK where he participated in the research of protocol optimization and media streaming.

**K. Liu** received his B.Eng. and Ph.D. degrees in Information Engineering from the Chinese University of Hong Kong, Shatin, Hong Kong, in 2008 and 2013, respectively. He is currently an Assistant Professor at the Advanced System Group under the Key Laboratory of Computer System and Architecture in the Institute of Computing Technology, Chinese Academy of Science, Beijing, China, where he participated in the research of protocol optimization and cloud computing.

**Jack Y. B. Lee** (M'95–SM'03) received his B.Eng. and Ph.D. degrees in information engineering from the Chinese University of Hong Kong, Shatin, Hong Kong, in 1993 and 1997, respectively. He is currently an Associate Professor in the Department of Information of the Chinese University of Hong Kong. His research group focuses on research in multimedia communications systems, mobile communications, protocols, and applications. He specializes in tackling research challenges arising from real-world systems. He works closely with the industry to uncover new research challenges and opportunities for new services and applications. Several of the systems research from his lab have been adopted and deployed by the industry, and is in daily use by millions of users.