Stochastic Exploration Congestion Control for Cellular Networks

Yasir Zaki
NYU Abu Dhabi

Thomas Pötsch
University of Bremen

Jay Chen
NYU Abu Dhabi

Lakshminarayanan Subramanian
New York University

Abstract

Legacy congestion controls including TCP and its variants are known to not perform well over cellular networks due to highly variable capacities over short time scales, self inflicted packet delays, and packet losses unrelated to congestion. Using measurements of 3G and LTE networks under high contention and mobility, we demonstrate the inherent unpredictability of the cellular channel and the bursty packet schedules observed by an end-host over short time scales; this channel unpredictability directly impacts the performance of stochastic model-driven protocols like Sprout. To handle these challenges, we present Verus, an end-to-end stochastic congestion control protocol that combines ideas from delay-based and loss-based congestion control to stochastically determine a good operating point without explicitly attempting to predict the cellular channel dynamics. Through a combination of simulations, empirical evaluations using traces, and real-world evaluations against standard TCP flavors, Cubic, and Sprout, we show that Verus outperforms these protocols in terms of throughput by 1.3x - 3.3x without incurring significant increases in delay (in comparison to Sprout).

1 Introduction

Cellular network channels are highly variable and users often experience fluctuations in their radio link rates over short time scales due to scarce radio resources making these channels hard to predict [24, 18, 6]. TCP and its variants are known to perform poorly over cellular networks due to high capacity variability, self-inflicted packet delays, stochastic packet losses that are not linked to congestion, and large bandwidth-delay products [13, 29, 30]. Recently, Winstein et. al. proposed Sprout [30], an end-to-end congestion control protocol based on stochastic modeling of the cellular channel with the dual objectives of achieving good throughput and significantly reducing self-inflicted queuing delays.

In this paper, we demonstrate using measurements on two commercial 3G and LTE networks that cellular network channels as observed at a single end-host granularity are hard to predict at short-time scales especially under high contention and mobility. We show four specific characteristics that have not been considered in detail in prior work. First, typical traffic characteristics observed at a receiver are highly bursty (even for smooth sending patterns) with variable burst sizes and burst interarrival periods which are sensitive to the level of contention. Second, standard linear predictors and k-step predictors perform poorly in predicting the bursty behavior of the channel over short time-periods. Third, while prior work has considered self-inflicted delay as a cause for high delay [30], we show that cross-competing traffic can play a significant role in end-to-end delay especially under high contention or when the cellular channel is near saturation. Fourth, device mobility has a substantial impact on channel characteristics that further compounds the performance and predictability challenges. The lack of predictability of the channel has important implications on the design of stochastic congestion control protocols especially when they attempt to explicitly model the channel behavior.

In this paper, we present Verus, a stochastic congestion control protocol that is primarily designed for highly variable channel conditions that are hard to predict. Without explicitly attempting to predict the cellular channel dynamics, the Verus protocol aims to be in constant stochastic exploration mode by tracking channel variations using cues from delay variations and packet losses to dynamically change its sending window. The key idea of the Verus protocol is to continuously learn a *delay profile curve* that captures the relationship between the sending window size and the perceived end-to-end delay. Using the delay profile curve and delay variation cues, the Verus protocol replaces the conventional Additive Increase (AI) in TCP with a combination of

stochastic increment/decrement steps to quickly react to rapidly changing channel conditions. While the stochastic control steps are primarily for congestion avoidance, Verus retains the loss-based multiplicative decrease step of TCP to enable quick reaction to congestion.

Verus uses measured end-to-end packet RTT from every ACK and the corresponding sending window at the time of packet transmission to form a (delay, Sending-Window) observation and approximates the delay curve from these observations through a spline interpolation curve fitting mechanism. Based on delay measurements over a sliding window, Verus uses a three step stochastic algorithm for window increment and decrement: first, it performs a stochastic increment of the sending window as long as the delay measurements do not increase beyond a threshold. Second, when Verus observes a delay buildup, it performs a stochastic decrement of the sending window. Finally, when Verus observes a packet loss, it performs a TCP-like multiplicative decrease. Verus uses slow start to learn the initial delay curve and updates this curve for every newly-received ACK.

We have implemented and tested Verus rigorously across a variety of environments including network simulations, controlled lab tests, and real world experiments. Our evaluations demonstrate that Verus achieves on the order of 1.3x - 3.3x greater throughput compared to Sprout, TCP Cubic, and TCP Vegas without significantly increasing the end-to-end delay. We also show that Verus performs well under mobility, and competing traffic, and that Verus is able to track and efficiently utilize channel changes to achieve higher throughput peaks.

2 Related Work

Legacy Congestion Control Protocols: Congestion control is an extensively studied topic with numerous variants of TCP. TCP Reno [5], TCP Tahoe [14] and TCP NewReno [12] are among the early popular variants which are loss-based and TCP Vegas [3] was among the earliest delay based control protocols. Most TCP implementations currently leverage TCP Cubic [11] which makes specific modifications to the increment function in conventional AIMD-based window control. There are also a number of other TCP flavors such as LEDBAT [25], TCP Nice [27], equation based rate control [8], and Binomial congestion control [2]. None of these legacy congestion control protocols are directly suited for cellular network conditions where the underlying channel changes at short time scales and the basic assumption that a link has a fixed capacity does not hold [28]. In addition, none of these TCP variants can distinguish stochastic losses that are part of the cellular environment from losses caused by congestion. Our work aims to combine ideas from conventional loss-based control with delay-based stochastic control drawing inspiration from protocols like Vegas [3].

Router-feedback Based Protocols: A common approach used in congestion control research to make TCP functional in new network environments (where TCP variants are not well suited) is to use router feedback in congestion control. Examples of such techniques include Explicit Congestion Notification (ECN) [7], VCP [31], or active queue management like RED [9], BLUE [4], CHOKe [23], AVQ [17], CoDel [20]. The problem with these methods is that they require modifications to intermediate routers which has remained a roadblock for adoption. In our setting, we aim to design an end-to-end congestion control protocol for cellular networks with no middle-box support or router feedback.

Recent Congestion Control Proposals: While TCP has remained a gold standard for several years, there have been several recent works on new congestion control protocols for new environments. Sprout [30] is a recent work specifically designed for the context of cellular networks. Sprout specifically focused on the problem of reducing self-inflicted delay that affects TCP and its variants under varying channel conditions. Sprout showed a significant reduction in the end-to-end delay experienced by flows in cellular networks while maintaining good throughput characteristics. We compare our work against Sprout [30] in detail later in this paper. Remy [29] focused on the problem of machine generated automated congestion control algorithms where a machine can be trained offline to learn congestion control schemes. The protocol designer specifies the desired targets of the network and Remy uses prior knowledge of the network to parametrize the protocol generation. Sivaraman et. al. [26] looked at the learnability of congestion control under imperfect knowledge of the network through an experimental study leveraging Remy as a design tool. Our work in contrast primarily focuses on performing congestion control in cellular networks where the underlying channel is hard to predict.

Another recent congestion control protocol is Data Center TCP (DCTCP) [1] that leverages ECN feedback to address several network impairments of TCP within data center networks. Recursive Cautious Congestion Control (RC3) [19] shows that the initial small window of TCP during slow start often wastes several RTTs until the flow fully utilize the available link bandwidth while RC3 uses several levels of lower priority services to achieve nearly optimal throughputs.

Cellular Performance: Several measurement studies have examined TCP performance problems in cellular networks. Cellular networks tend to over-dimension their buffers by using large buffers at base stations to smooth the overall flow of traffic. As a result, conventional congestion control protocols result in "bufferbloats" [10]

and multi-seconds delays. Jiang et. al. [16] have shown the severity of bufferbloats through extensive measurements done on 3G/4G commercial networks. The authors of [21] run long-term measurements to investigate the end-to-end performance of mobile devices within and across a wide range of carriers (i.e., 100), using 11 cellular network access technologies. The results show that there are significant performance differences across carriers, access technologies, geographic regions and time. Zaki et al. [32] conclude similar observations on cellular networks in developing regions as well as developed regions. Huang et al. [13] studied the effect of network protocols on 3G and 4G LTE networks by means of active and passive measurements. They discovered that TCP connections over LTE have various inefficiencies such as slow start. In comparison to 3G networks, LTE shows lower delays while many TCP connections ($\sim$52%) under-utilize the available bandwidth of LTE.

3 Cellular Channel Contention & Mobility

In this section, we use measurements on two commercial 3G and LTE networks to demonstrate the high variability of the cellular channel especially under contention and mobility and demonstrate the need for an exploration based protocol for such environments. We make four specific observations.

1. **Competing traffic:** Prior work on Sprout [30] explicitly did not consider competing traffic based on the observation that the cellular scheduler maintains separate queues for each user. Despite this queue-level isolation, flows still compete for the same radio resources. When two flows contend for radio resources and their sending rates approach network capacity, we observe that cross-competing traffic can play an important role in end-to-end delay.
2. **Burst scheduling:** The typical traffic characteristics observed at a receiver are highly bursty with variable burst sizes and burst inter-arrival periods which are sensitive to the level of contention.
3. **Cellular channel prediction is hard:** Cellular channels fluctuate rapidly over short time scales and simple channel predictors based on moving samples have a difficult time tracking these variations. This demonstrates the need for a highly adaptive exploration based protocol which may not perfectly track the channel variations.
4. **Mobility increases channel variability:** Mobility causes higher delays and larger fluctuations in the channel.

3.1 Measurement Setup

We have measured cellular network performance under different conditions, testing the effects of several factors including: data rates, mobility, competing traffic, and 3G or LTE. The measurements were conducted on two commercial cellular networks[1] for both downlink and uplink. Our measurement setup consisted of a rack server with a global IP address and a laptop. The client laptop was tethered to a Sony Xperia Z1 LTE mobile phone. We implemented our own measurement tool that sends/receives UDP packets between the client and the server at specified sending intervals. We tagged packets with sequence numbers and included the sender timestamp to calculate the one-way delay at the receiver.

Table 1: Downlink competing traffic configurations

	User #1 (UE#1) rate	User #2 (UE#2) rate
Scenario 1	1 Mbps	10 Mbps; active during the 1st, 3rd and 5th minute
Scenario 2	5 Mbps	10 Mbps; active during the 1st, 3rd and 5th minute
Scenario 3	10 Mbps	10 Mbps; active during the 1st, 3rd and 5th minute

Table 2: Measurements parameters/scenarios

Scenario	Pkt size (Bytes)	Pkt IAT (ms)	Rate (Mbps)
Etisalat LTE DL static LTE DL mobile	500 1000 1250	8, 4 & 0.4 16, 8 & 0.8 20, 10 & 1	0.5, 1 & 10 0.5, 1 & 10 0.5, 1 & 10
Etisalat 3G DL/UL	500	8, 4 & 0.4	0.5, 1 & 10
Du 3G/LTE DL static & mobile DL mobile	500	0.4	10

We used Network Time Protocol (NTP) at the beginning of each experiment to synchronize the clocks of the client and the server. NTP has multi-millisecond accuracy over wireless networks, thus the delay of each measurement is subtracted by the minimum delay seen through the measurement time, i.e., the propagation delay to account for the slight difference between the two clocks. We assume that the minimum delay is equal to the propagation delay, and therefore the delay represents the buffering delay seen by the packet. Several different measurements were conducted: stationary, mobile with low speed, and stationary with competing traffic. Table 1 and 2 summarize the measurement sets; each measurement was executed for five minutes.

[1]Etisalat and Du are the UAE's cellular providers and they provide LTE coverage in most parts of the major cities.

3.2 Competing traffic

While cellular networks maintain separate queues for flows, competing traffic can directly impact end-to-end delay characteristics especially when the network operates close to capacity even over short time-scales. Flows compete for the same radio resources and under high contention or when operating close to network capacity, the radio scheduler does impact end-to-end delay characteristics. Prior work by Sprout [30] primarily focuses on self-inflicted delay and does not consider the impact of competing traffic due to the queue isolation assumption. Using a simple measurement, we show that competing traffic, when operating close to network capacity, can be an important cause for large end-to-end delay fluctuations observed by flows. We consider two users competing at the same cellular base station such that when both users are active, their combined data rates is almost equal to the 3G capacity limit. The first user is constantly transmitting at a fixed rate while the second user is set to operate in ON/OFF periods of one minute intervals as summarized in Table 1. Figure 1 shows the packet delays for the first user when the second user is ON/OFF. We observe that during the non-competing periods, the average delay is quite low. However, when the second user is ON the average packet delay for the second user increases. This increase is especially high when the combined data rate reaches the capacity.

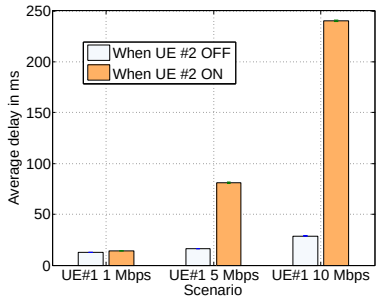


Figure 1: User #1 average packet delay with/without competing traffic on 3G

3.3 Packets experience burst scheduling

The packet arrival times at the receivers exhibit a cellular radio scheduler phenomena known as "burst scheduling". The radio scheduler serves users at different one millisecond Transmission Time Intervals (TTI), and the amount of data sent during the serving TTI is determined by the experienced radio conditions, which leads to the sending of a burst of several packets. Figure 2 illustrates this bursty arrival at the receiver for one of our LTE 10 Mbps downlink measurements. Since we used a constant sending rate in the experiment, the packets were sent with an inter packet send time of 0.4 milliseconds, which can be seen between the dots of each burst.

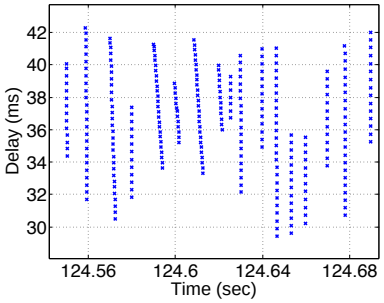
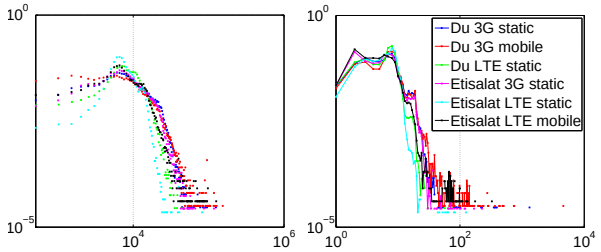


Figure 2: LTE DL 10 Mbps burst arrival time

To take into account this bursty behavior of cellular radio schedulers, we separate out packet interarrival time into the inter *burst* arrival time and the burst size. Figure 3 shows the probability density function of the burst size in bytes and the distribution for inter burst arrival time for different scenarios within the Etisalat and Du measurements: 3G, LTE, 3G with mobility, and LTE with mobility. We observe that the burst size and inter burst arrival time are difficult to predict and change significantly depending on the scenario. These distributions do not map easily to a Poisson process as was assumed in Sprout [30].



(a) Burst size (b) Burst Interarrival time

Figure 3: DL burst probability distribution function

3.4 Channel prediction is hard

To demonstrate that the channel is non-trivial to predict, we used simple predictors to compare the predicted data with the measured data on a 3G downlink with one user receiving at 10 Mbps. The prediction is performed over the received data in time windows of 20 ms. Figure 4 shows the results of two predictors: a Linear predictor, and a k-step ahead predictor using Matlab's armax model. These predictors fail to track the high variations of the channel. While one can experiment with a variety of other predictors, the primary take-away message is that standard prediction mechanisms even using the most recent samples are far from capturing the bursty behavior of the end-to-end channel.

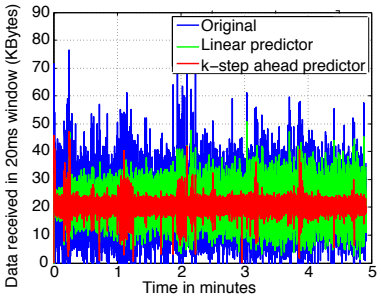
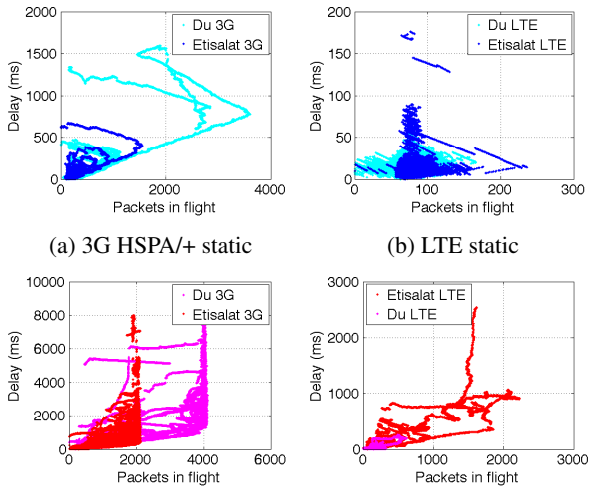


Figure 4: 3G DL 10 Mbps received data prediction

3.5 Mobility increases channel variability

To highlight the effect of mobility on channel variability, we compare two settings: static setting (device in fixed location) and a mobile setting (device location changing with time). In both settings, we examined the relationship betwen delay and packets in flight under a constant packet transmission rate. Figures 5a and 5b illustrate a scatter plot of these two parameters for 3G and LTE networks under the static setting and Figures 5c and 5d show the mobile setting. The results show that the channel is significantly more variable for the mobile setting than the static scenario. We also observe an increase in the variability of the size of the bursts under high mobility than the static setting (illustrated by traces of points along a line). We observe similar results on the mobile uplink.



(a) 3G HSPA/+ static

(b) LTE static

(c) 3G HSPA/+ mobile

(d) LTE mobile

Figure 5: Downlink delay vs. packets in flight

4 The Verus Protocol

Motivated by our measurements, we designed Verus, an end-to-end congestion control protocol that does not make any assumptions about the channel or try to model it. Verus is a sender-driven protocol that stochastically explores the channel. It is both a congestion control and congestion avoidance protocol for highly fluctuating channels like cellular networks. The key idea behind Verus is to use a delay profile curve which captures the relationship between the estimated RTT and the sending window.

Verus is a three stage protocol: stochastic increment, stochastic decrement, and multiplicative decrease. Verus leverages the basic TCP increment and decrement functions of TCP, but changes the way increment and decrement are applied. Verus uses a similar multiplicative decrease mechanism as TCP to deal with packet losses and timeouts, However, Verus enforces stochastic increment and decrement through the use of a delay profile curve.

Verus estimates the packet RTT from the stochastic steps, and uses the delay profile curve to compute the sending window. Verus maintains a sending window over a period equal to the estimate of the RTT. However, the actual sending of packets happens in smaller epochs of ϵ duration, for example 10 ms. In this manner, the sender is able to react quickly to sudden channel changes and can track the temporal changes of the channel. Figure 6 shows the Verus time framework.

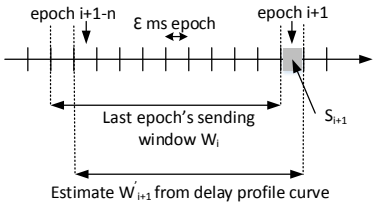


Figure 6: Verus time framework

The Verus protocol is implemented at the end points — the sender and the receiver. The sender encompasses all main protocol features, whereas the receiver is responsible for acknowledging the received packets.

4.1 Verus Initialization (Slow Start)

The first phase of the Verus protocol is the slow start phase. The slow start phase is very similar to TCP's slow start; where the sender starts sending with a sending window W_i equals to one and for every received acknowledgment the sender increases the window by one. Slow start is used by Verus to build the delay profile curve used to estimate the next sending window W'_{i+1}.

The Verus slow start phase has two conditions to exit:

1. The first is defined as encountering a packet loss. Packet losses can be identified by missing sequence numbers (identified through the received acknowledgments). However, out of sequence packets is a

normal practice in cellular networks, thus loss detection cannot rely solely upon identifying a gap in received sequence numbers. We use a simple missing sequence numbers buffer with a certain timeout timer, where for each missing sequence number a timeout timer is set, and if the packet comes within the set timer then the timeout timer is stopped. In case the timer expires before the packet arrives, then the packet is treated as lost.

2. The second exit condition is defined to be a timeout event where during a timeout all packets within the network are treated as lost; this could happen if there is a tail drop buffering point in the network.

After exiting the slow start phase the first delay profile curve is built.

4.2 Delay Profile Curve

One of the main components of Verus is the creation and maintenance of a delay profile curve. The delay profile curve consists of data points with x-coordinate corresponding to the sending window (W_i) and y-coordinate corresponding to the packet RTT delay ($D_{i,p}$). For every received ACK'ed packet at the sender, the corresponding data point in the delay profile curve is updated. The update is done using an Exponentially Weighted Moving Average (EWMA) function to track the channel history. Figure 7 shows an example of a delay profile curve.

Verus bases its decision based on the RTT feedback from the receiver. The receiver sends an acknowledgment back to the sender upon the arrival of every packet. The sender estimates the round trip delay by subtracting the sending time-stamp of that packet from the receiving time. The round trip delay experienced by a packet p received in epoch i is denoted as $D_{i,p}$, and the maximum experienced delay per epoch i is denoted as $D_{max,i}$.

Every packet p is sent as part of a sending window W_i, i.e., number of packets in flight. W_i represents the sending window to be maintained over an estimate of the RTT and denotes the number of MTU sized packets to be sent. The sender keeps track of W_i for each sent packet and thus upon receiving a packet p, the sender will obtain a pair of $D_{i,p}$ and W_i. This way the sender can construct a delay versus sending window curve profile through cubic spline regression. Figure 7 shows an example of the slow start delay points (green dots) and the creation of the delay profile curve upon slow start exit (red curve).

The purpose of the delay profile curve is to estimate what the next upcoming sending window for epoch $(i+1)$ should be. We denote this sending window as W'_{i+1}, i.e., the sliding sending window that is evolved by an epoch (see Figure 6).

The delay curve is upper bounded by a maximum tolerable delay $D_{thr}(t)$ and is calculated as in equation 1. Using $D_{thr}(t)$ with the delay curve, a maximum sending window can be estimated and is denoted as $W_{thr}(t)$.

$$D_{thr}(t) = \overline{D(t)} + 3 \cdot D_{dev}(t) \quad (1)$$

where, $\overline{D(t)}$ is the EWMA of the ACK'ed packet delays (explained in next subsection). $D_{dev}(t)$ is the EWMA (25% weighted) of the standard deviation of D_p and D_{p-1}.

4.2.1 Delay Curve Updates

Due to the dynamic nature of the channel and the changes in the radio environment, the delay curve that Verus uses has to be updated according to the current network state to track the changes of the channel capacity.

The delay profile curve is maintained and updated at the sender. For each received ACK the delay profile curve delay value of the ACK'ed packet sending window is updated using an EWMA function. Then, a new delay profile curve is generated using the cubic spline regression method. The update of the curve is done regularly after a couple of epochs have elapsed, i.e., every second.

4.3 Sending Window Estimation

The next step is to estimate stochastically how many packets to send over the next epoch ε. There are two main time slots within the protocol structure: First, a large sliding window that is equal to an estimate of the round trip delay, denoted as $\overline{D(t)}$. $\overline{D(t)}$ is an EWMA of the round trip delays experienced by all ACK'ed packets, i.e., EWMA (87.5% weighted) of D_p and D_{p-1}. The second slot is a smaller one called epoch (ε) with a length of, e.g., 10 ms.

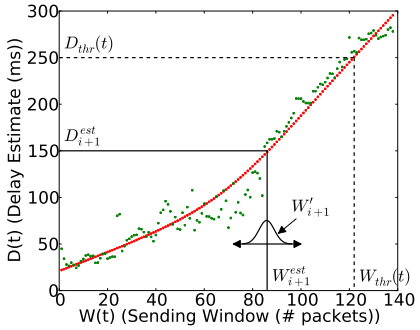


Figure 7: Verus delay vs. W_i profile curve

The sending window W_i is maintained at the larger sliding window, whereas S_i represents the data sent during the i^{th} epoch. The target of Verus is to calculate S_{i+1}, which is the data to be sent in the future $(i+1)$ epoch

while maintaining W_i over $\overline{D(t)}$. Within an epoch, W_i is incremented with every packet sent and decremented with every ACK received.

At the beginning of epoch $i+1$, the protocol calculates how many packets it has to send so as to achieve a certain sending window. This is calculated using equation 2.

$$S_{i+1} = max[0, (W'_{i+1} + \frac{2-n}{n-1} \cdot W_i)] \quad \text{with } n = \left\lceil \frac{\overline{D(t_i)}}{\varepsilon} \right\rceil \tag{2}$$

where W'_{i+1} is the estimated sending window to be maintained at epoch $i+1$ in the future, which is stochastically estimated from the delay profile curve, W_i is the actual sending window at the end of epoch i (i.e., the sending window at the time before making the next epoch decision), and n is the number of epochs in the estimated average round trip delay $\overline{D(t_i)}$ with t_i being epoch $i+1$ absolute starting time.

4.4 Stochastic Increment and Decrement

For each epoch ε, the sender keeps track of the maximum round trip delay $D_{max,i}$, which is the maximum delay of all ACK'ed packets received within that epoch. If there were no packets ACK'ed during the epoch then $D_{max,i}$ is calculated using equation 3.

$$D_{max,i} = max[D_{max,i-1}, \text{delay of smallest unACK'ed Pkt}] \tag{3}$$

The smallest un-ACK'ed packet is the packet with the sequence number that is just after the sequence number of the last received ACK'ed packet. The reason behind this is to calculate the delay of the next possible ACK that might arrive.

At the end of each epoch, $D_{max,i-1}$ is updated with a 20% weighted EWMA of $D_{max,i}$. The difference between $D_{max,i}$ and $D_{max,i-1}$ is denoted as ΔD_i and represents the relative increase/decrease of the maximum delay experienced between the last two epochs. $\overline{\Delta D_i}$ gives an estimate of whether the delay is increasing or decreasing. Based on this information, an estimate on the next epoch's maximum delay is calculated as follows:

$$D_{est,i+1} = \begin{cases} max[D_{min}, (D_{est,i} - \delta_1)] & \text{if } \frac{D_{max,i-1}}{D_{min}} > 6 \\ max[D_{min}, (D_{est,i} - \delta_2)] & \text{if } \overline{\Delta D_i} > 0 \\ D_{est,i} + \delta_1 & \text{otherwise} \end{cases} \tag{4}$$

where D_{min} is the minimum delay experienced by Verus and the δ's are the delay reduction parameter for the next epoch. The values of δ_1 and δ_2 are constant value that are set to 2 ms and 1 ms, respectively. Depending upon the relative increase or decrease of the delay, Verus will either decide to increase the delay or decrease it. There is one exception in the equation which puts an upper bound on the delay not to increase beyond a certain threshold. The threshold is set in a way that the delay should not increase above six times the minimum RTT experienced by the network.

By using $D_{est,i+1}$ with the delay profile curve (Figure 7), the corresponding sending window $W_{est,i+1}$ is found. W'_{i+1} is stochastically calculated next as follows:

$$W'_{i+1} = \begin{cases} \text{Normal\_dist}(W_{est,i+1}, \sigma) & \text{if } \overline{\Delta D_i} < 0 \\ W_{est,i+1} & \text{otherwise} \end{cases} \tag{5}$$

The sending window in case of increase is stochastically drawn from a normal distribution with the mean equals to $W_{est,i+1}$ and the variance σ that is a function of how far $W_{est,i+1}$ is from the threshold $W_{thr}(t)$. σ is calculated as in equation 6. In case of a decrease, the sending window is just taken from the delay profile curve.

$$\sigma = \frac{|W_{thr}(t) - W_{est,i+1}|}{k} \qquad 2 < k \leqslant 4 \tag{6}$$

4.5 Verus Loss Handling

Packet losses are handled using the following procedure: if a packet loss is detected, Verus reduces the sending window as in equation 7 and calculates the new $D_{est,i+1}$ from the delay profile curve based on the new W_{i+1}.

$$W_{i+1} = M \cdot W_{loss} \tag{7}$$

where, W_{loss} is the lost packet sending window, M is the multiplicative decrease factor (set to 70%), and W_{i+1} is the next epoch's sending window after the loss.

The reason why we choose the sending window of the lost packet is because this sending window was responsible for the packet loss and there is a delay in the feedback loop until the loss is identified where at this stage the sending window can be increased from the original window of the packet loss.

Once the packet loss is identified and the sending window has been multiplicatively decreased, Verus enters a loss phase. During the loss phase, the delay profile curve is not updated until the loss phase is over. The loss phase is important because Verus bases its sending decision upon the delay experienced by the network. When a loss happens, incoming packets that arrive after the loss would have a much lower buffering delay that will not reflect the real network delay.

During the loss phase and upon receiving an ACK packet, the sending window W_{i+1} is inflated by $1/(W_{i+1})$

(similar to TCP) and the corresponding delay estimate $D_{est,i+1}$ is calculated from the delay profile curve.

Verus exits the loss phase once ACKs of packets sent after the loss are received, i.e., if the protocol receives an ACK with sending window that is smaller than or equal to the current sending window.

In addition to packet losses, Verus also implements a timeout mechanism to be used in case all packets are lost. For every received ACK, a timeout trigger is set for the next ACK to come. The trigger is set to be equal to three times the last received delay. If no ACK arrives before this time then a timeout will be triggered.

During a timeout, Verus resets the current sending window W_i to zero, sets the estimate of the delay to be used by the next epoch $D_{est,i+1}$ to D_{min}, and triggers the Verus slow start phase.

5 Implementation and Evaluation

Our implementation of Verus consists of a sender and receiver application, both written in C++. The sender application runs in a multi-threaded environment and uses the real time extension library *librt*. In addition, the sender application uses the cubic spline interpolation from the *ALGLB* library to compute the delay curve. Upon sending a packet, the sender includes a time-stamp in the payload of the PDU which is echoed back by the receiver by means of an acknowledgment.

In this section, we evaluate the performance of the Verus protocol by comparing it to other transport congestion control protocols: Sprout, TCP Cubic and TCP Vegas. First, we compare Verus to Sprout in a controlled lab environment using Cellsim. This experiment is used as our baseline to compare the performance of the protocols in Sprout's experimental environment using Sprout's channel traces. Cellsim has the limitation that it can only evaluate a single client without contention, thus we have implemented the Verus protocol in a simulation environment to be able to evaluate how it behaves under contention. We continue our evaluation on real cellular networks to test and evaluate the protocols in real scenarios ranging from 3G, LTE, to multiple competing flows.

5.1 Cellsim Baseline

To compare with Sprout, we use the Cellsim [30] network emulator used in the Sprout paper. The Cellsim setup consists of three PCs at which the intermediate PC receives packets on two Ethernet interfaces, delays these packets according to the channel trace and then sends them to the second network interface. We slightly modified the Cellsim code by adding a limited buffer on both downlink and uplink direction to reflect a more realistic evaluation scenario, since real cellular networks have limited buffers.

We compare the performance of Verus using the Cellsim environment with Sprout, TCP Cubic, and TCP Vegas. We used one client and one server as end-points with another machine running Cellsim emulating the cellular channel between the end-points. Four different trace files are used to emulate different 3G/4G cellular commercial networks. The trace files used in the evaluations were collected by [30] from four different US cellular operators, namely: 3G AT&T, LTE T-Mobile, 3G Verizon and LTE Verizon. Figure 8 shows the time series of the throughput and the delay of Verus, Sprout, TCP Cubic and TCP Vegas using the LTE Verizon trace. As expected, TCP Cubic achieves the highest throughput compared to the other protocols but it also incurs the highest delay. Sprout achieves the smallest delay and the smallest throughput. Verus achieves almost the same throughput as TCP Cubic without compromising the delay significantly over Sprout.

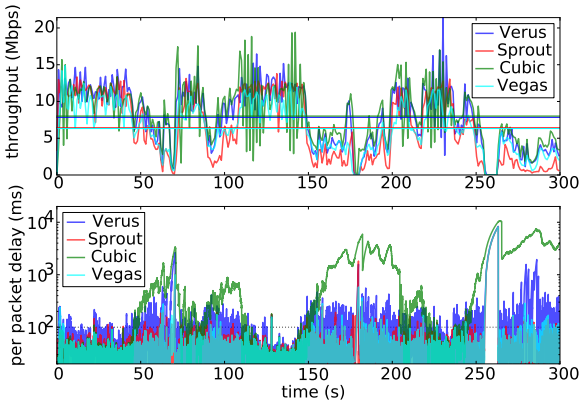


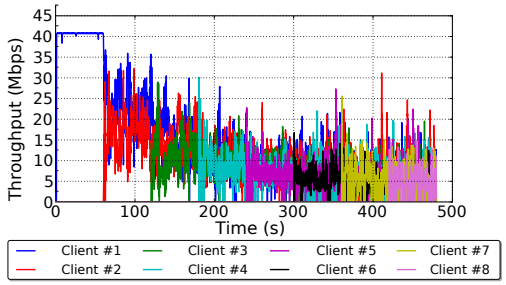
Figure 8: Verizon LTE DL performance comparison

The rest of the results are summarized in Table 3. These results are averaged over three runs for each experiment (i.e., each with a different start time of the trace). The red highlighted cells in the table represent the places where Verus under-performs compared to the other three protocols. For example in LTE AT&T, Verus experiences higher average delays compared to Sprout. However, it achieves 20% higher throughput compared to Sprout. Verus always has higher throughput than Sprout with the penalty of a slightly increased delay.

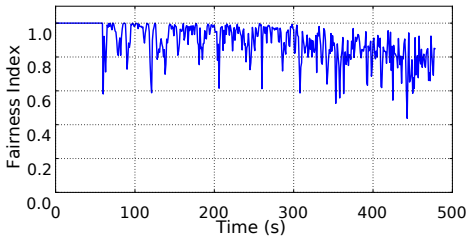
As we have shown in Section 3, competing traffic affects the performance depending on the amount of data sent. Cellsim is designed to emulate only a single client and single server. The drawback of Cellsim is that it cannot emulate competing traffic from several clients on the channel itself since it does not have a radio scheduler. As a result, even if two traces are used for two clients the effect of radio scheduling and competition will not

Table 3: Cellsim results with Verus average throughput improvement and delay reduction over Sprout, TCP Cubic and Vegas

Protocol	Verus average throughput improvement	Verus average delay reduction
3G T-Mobile		
Verus	1.0x	1.0x (0.255 s)
Sprout	2.5x	0.8x (0.202 s)
Cubic	0.8x	18.2x (4.64 s)
Vegas	1.1x	0.7x (0.17 s)
3G Verizon		
Verus	1.0x	1.0x (0.221 s)
Sprout	1.8x	0.9x (0.188 s)
Cubic	0.7x	47.7x (10.544 s)
Vegas	1.0x	0.8x (0.186 s)
LTE Verizon		
Verus	1.0x	1.0x (0.176 s)
Sprout	1.1x	0.7x (0.119 s)
Cubic	0.9x	3.7x (0.655 s)
Vegas	1.1x	0.6x (0.112 s)
LTE AT&T		
Verus	1.0x	1.0x (0.156 s)
Sprout	1.2x	0.7x (0.108 s)
Cubic	0.4x	9x (1.4 s)
Vegas	1.3x	0.8x (0.118 s)



(a) Throughput



(b) Jain's fairness index

Figure 9: Verus wired scenario fairness

be reflected.

5.2 Simulations

Due to Cellsim's limitations and because we wish to study the effect of competing traffic, we have implemented Verus in an LTE simulation environment using the OPNET simulator [22]. We ported the code into OPNET to experiment with Verus first in a controlled environment. The simulation environment allows us to run several scenarios over different access links such as wired Ethernet and cellular LTE. For these simulations, we did not compare Verus to Sprout since it required porting its code into the OPNET environment. We return to Sprout in our real-world comparisons in Section 5.3.

5.2.1 Competing Traffic (Wired)

We begin by evaluating the throughput and fairness of eight competing Verus flows (clients) all sharing a bottleneck wired Ethernet link of 40 Mbps capacity. The simulation setup is configured so that in every 60 seconds a new Verus flow starts sending data, thus increasing the number of competing flows over time. The Verus flows use a full buffer File Transfer Protocol (FTP) traffic model. The bottleneck link is configured with a tail drop buffer management in order to have a finite buffer and realistic packet losses. Our evaluation here highlights the fairness of the Verus protocol under contention.

Figure 9a shows the throughput of the eight Verus clients over time. In the first 60 seconds when only one Verus flow is active, the flow is fully utilizing the 40 Mbps bottleneck link. Then, a second Verus flow joins the network and it can be noticed that the two flows alternate around the bottleneck capacity, each achieving an average throughput of 20 Mbps. Figure 9b shows the Jain's fairness index calculated as a moving window of one second. The results shows that although Verus is a stochastic protocol, it still achieves a good stochastic fairness of about 80% over 1 second windows when all eight Verus clients are active and competing over the bottleneck link. The Jain's fairness index defined in [15] as:

$$f(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} {x_i}^2} \tag{8}$$

where x_i is the normalized throughput of the i-th user and n is the number of clients. The fairness index always ranges from zero to one, where $\frac{1}{n}$ represents the worst case and one the best case. Hence a fairness index of 0.8 over short time scales of 1 second, represents a high value for a stochastic exploration protocol which can quickly change its congestion window over short time scales. Over longer time scales, Verus has a fairness index of close to 1.

5.2.2 Competing Traffic (LTE)

Next, we evaluate Verus under high contention in an LTE cellular network. The simulation is configured with 20 Verus clients all served by the same LTE base-station. The clients are randomly distributed within the base-station coverage area and are configured with a FTP full

buffer traffic model. Two different scenarios are evaluated: pedestrian and vehicular. In the pedestrian scenario all 20 Verus clients are configured with a speed of 3 kmph, whereas in the vehicular case the users are configured with a speed of 120 kmph. In this evaluation, Verus is compared against TCP Cubic (with Linux 2.6 configuration) and TCP New Reno (with Windows 7 configuration). Figure 10 shows the results of the comparison, where the x-coordinates represents the average downlink end-to-end delay and the y-coordinates represents the average throughput. Each dot in the figure corresponds to the results of one of the 20 Verus clients.

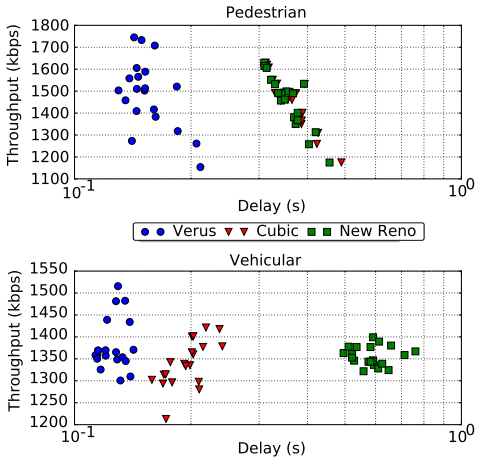
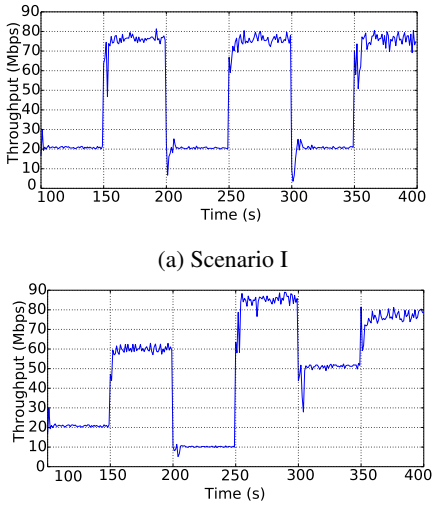


Figure 10: Simulation results of 20 competing clients

We see from the results that Verus, represented by the blue circles, achieves the best performance in both scenarios, i.e., pedestrian and vehicular compared to TCP cubic, represented by the red triangles, and TCP Vegas, represented by the green squares. Verus achieves better performance in terms of higher throughput and lower delay. The 20 Verus users individual results vary due to the fact that the users experience different channel conditions due to their initial positions and mobility. In the pedestrian results (upper figure), both TCP flavors achieve similar results, whereas in the vehicular results TCP New Reno suffers from higher delays due to higher channel fluctuations caused by the vehicular speed. The main takeaway from this analysis is that Verus outperforms classic TCP congestion control protocols in the LTE environment, especially under high contention and mobility.

5.2.3 Verus Adaptation Evaluation

Next, we wanted to measure how quickly Verus can adapt to high channel variations. The simulation setup is configured in a way that the link capacity changes suddenly. We consider two highly variable link settings. In the first setting, we consider a link with alternating capacity every 50 seconds between 20 Mbps and 80 Mbps. The second is configured to change to different link capacities every 50 seconds (20, 60, 10, 90, 50 and 80 Mbps). Figure 11 shows Verus' throughput over time for the two scenarios. This demonstrates that the stochastic nature of the Verus protocol can handle sudden changes and that it adapts very well and quickly to varying capacity in both scenarios.



(a) Scenario I

(b) Scenario II

Figure 11: Varying link capacity evaluations

5.2.4 Delay Profile Curve Characteristics

One of the crucial components of Verus is the delay profile curve. A delay profile curve notion of fairness is: two competing flows with similar (variable) network characteristics should converge to a similar delay profile curve. To investigate this, we setup an experiment with five Verus flows within the LTE cellular network; each is configured with a full FTP buffer traffic model.

We started with a stationary scenario, where the five competing Verus flows were placed in the same location. This means that these flows would have similar channel conditions. We gathered the delay profile curves for all five flows, where these curves are generated every second. We ran the simulation for 1000 seconds. From these curves we found they are nearly identical and do not change significantly over time.

We considered two different mobile scenarios: one where all five flows move exactly on the same path within the LTE cell and one with random mobility. In the first scenario, we find that the delay curves of all five Verus flows converge in the same general region because all flows experience similar channel conditions. In the

second scenario, the flows experience different instantaneous channel conditions. However, all flows have the same stochastic channel after a long-term simulation run. Although, the delay curves of the five flows at specific points in time are different, they all come from the same population as can be seen in Figure 12.

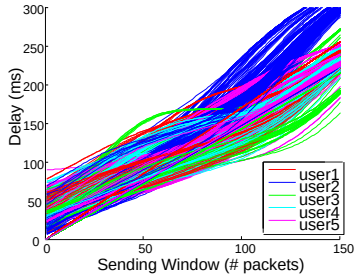


Figure 12: Delay curves for five mobile Verus flows

In order to confirm the assumption of the same stochastic channel after a long simulation run, we compare the probability distribution function (PDF) of the sending windows of each Verus flow. These distributions are drawn from the delay curve for each individual Verus flow in the mobile experiment. Figure 13 shows the PDF of the sending windows for the different flows, and the results confirm that they have similar distributions.

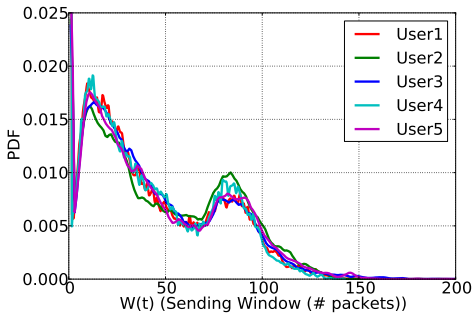


Figure 13: Sending window PDF for five mobile Verus flows

The two peaks in the Figure 13 at sending window of 13 packets and the other at sending window of 85 packets come from the delay estimation process of the Verus protocol. When the RTT measurements of the network are not increasing, Verus increases its sending window stochastically using the delay curve until receiving indication that the RTT of the network is increasing. Then Verus would uses stochastic decrement to reduce the sending window. The protocol alternates between these two phases and the upper/lower sending windows that it reached are reflected by the distribution peaks. The remaining portions of the distribution are Verus oscillating between these phases.

5.3 Real Network Evaluation

Here, we present the results of real 3G and LTE cellular networks experiments. The idea behind these experiments is to test the effect of competing traffic on Verus, Sprout, TCP Cubic, and TCP Vegas performance on real networks. The setup of these experiments are as follows: three laptops are tethered to three LTE mobile cell phones (i.e. 2x Samsung Galaxy S4 and 1x Sony Xperia Z1), all running three simultaneous flows. All devices communicate with a server with a public IP address that is set up at our university campus. In order to investigate competing traffic of Verus itself and against other congestion control protocols, several evaluations are performed which can be summarized as follows:

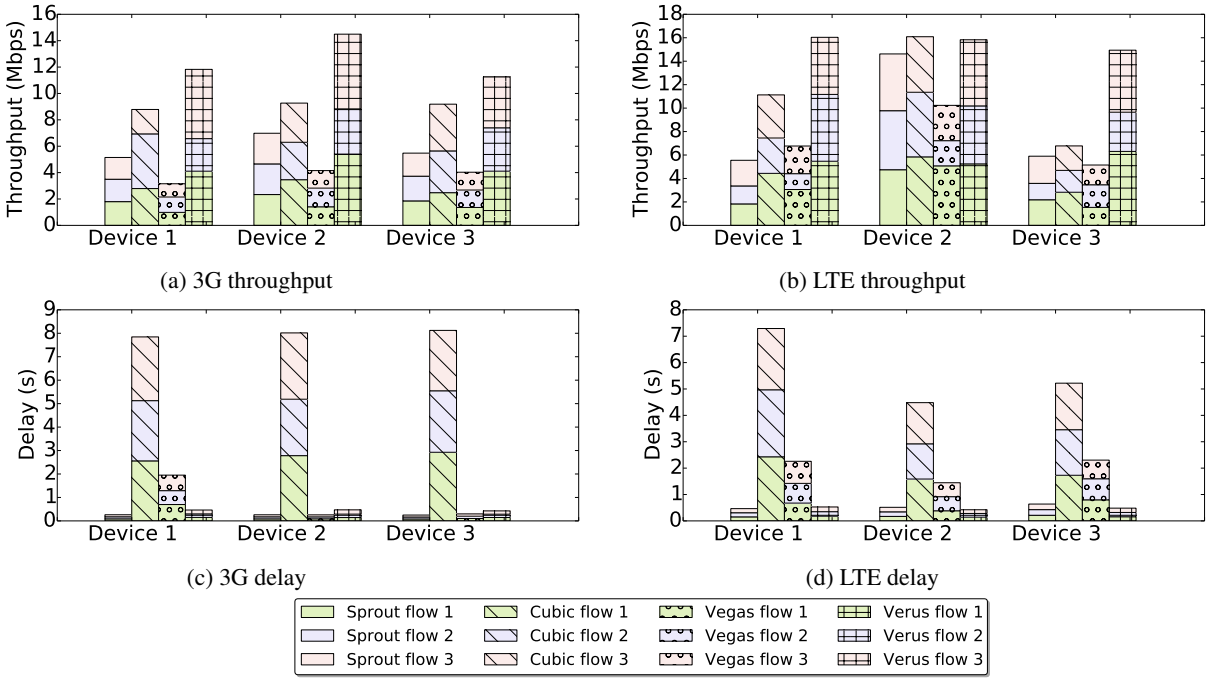
1. Three phones each running three Verus flows
2. Three phones each running three Sprout flows
3. Three phones each running three Cubic flows (secure copy (scp) download)
4. Three phones each running three Vegas flows (secure copy (scp) download)

All experiments have been performed on Etisalat's 3G and LTE network on fixed locations, i.e. without mobility. The duration of each run was two minutes and has been repeated five times. All tests were done on the downlink.

Figure 14 shows the averaged values (per flow) of the five experiments for Verus, Sprout, TCP Cubic and TCP Vegas. Each group of bars denotes the mean values run per device. The figures show substantial differences among the three devices. Verus shows nearly an equally distributed share of the capacity, whereas the throughput of Sprout and TCP Vegas differ across the devices. We confirmed through further experiments that this effect is mainly caused by contention and the design of the protocols and is not related to the physical equipment. The figures also show that Verus outperforms nearly all other protocols in terms of throughput. From the delay figures it can be seen that in both networks (3G and LTE), Verus and Sprout show similar values. On LTE, Verus even shows the lowest delay. The TCP protocols (Cubic and Vegas) on the other hand show much higher delays reaching even up to 7-8 seconds for Cubic.

6 Conclusions and Future Work

In this paper, we presented Verus, a stochastic exploration congestion control protocol that is tailored for cellular networks. Through stochastic exploration using delay measurements and a delay profile curve, Verus is able to track the cellular changes on the channel and adapts to the changing conditions and to competing traffic. We



(a) 3G throughput

(b) LTE throughput

(c) 3G delay

(d) LTE delay

Figure 14: Averaged throughput and delay values for each flow over five experiments. Per device, from left to right: Sprout, TCP Cubic, TCP Vegas, and Verus

tested Verus under a variety of experimental scenarios ranging from controlled lab setup using the Cellsim tool, OPNET simulation environment, and real world tests over 3G and LTE networks. Our evaluations have shown that Verus achieves high throughput comparable to TCP Cubic while outperforming Sprout and other legacy TCP variants. In comparison to Sprout, Verus incurs only a marginal increase in the overall RTT while TCP Cubic incurs a large self-inflicted delay. Verus smoothly handles competing traffic on high contention and also quickly adapts to channel variations. There are a number of sensitivity parameters that have been fixed for the protocol and can be tuned to achieve better performance. In our future work, we plan to automatically tune these parameters. We also plan to implement Verus as a native transport protocol and test it with a number of real world applications.

References

[1] ALIZADEH, M., GREENBERG, A., MALTZ, D. A., PADHYE, J., PATEL, P., PRABHAKAR, B., SENGUPTA, S., AND SRIDHARAN, M. DCTCP: Efficient Packet Transport for the Commoditized Data Center. ACM SIGCOMM.

[2] BANSAL, D., AND BALAKRISHNAN, H. Binomial congestion control algorithms. In *Proceedings of 20th Conference of the IEEE Computer and Communications Societies. INFOCOM 2001.* (2001), vol. 2.

[3] BRAKMO, L. S., O'MALLEY, S. W., AND PETERSON, L. L. *TCP Vegas: New techniques for congestion detection and avoidance*, vol. 24. ACM, 1994.

[4] CHANG FENG, W., SHIN, K. G., KANDLUR, D. D., AND SAHA, D. The BLUE active queue management algorithms. *IEEE/ACM TRANS. NETWORKING* (2002), 513–528.

[5] COX, D., AND DEPENDENCE, L.-R. a review. *Statistics: An Appraisal, HA David and HT David (Eds.), The Iowa State University Press, Ames, Iowa* (1984), 55–74.

[6] EKMAN, T. *Prediction of Mobile Radio Channels - Modeling and Design*. PhD thesis, Signals and Systems, Uppsala University, Sweden, 2002.

[7] FLOYD, S. TCP and Explicit Congestion Notification. *SIGCOMM Comput. Commun. Rev. 24*, 5 (Oct. 1994).

[8] FLOYD, S., HANDLEY, M., PADHYE, J., AND WIDMER, J. *Equation-based congestion control for unicast applications*, vol. 30. ACM, 2000.

[9] FLOYD, S., AND JACOBSON, V. Random early detection gateways for congestion avoidance. *Networking, IEEE/ACM Transactions on 1*, 4 (Aug 1993).

[10] GETTYS, J. Bufferbloat: Dark Buffers in the Internet. *Internet Computing, IEEE 15*, 3 (May 2011).

[11] HA, S., RHEE, I., AND XU, L. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS Oper. Syst. Rev. 42*, 5 (July 2008).

[12] HENDERSON, T., FLOYD, S., GURTOV, A., AND NISHIDA, Y. The newreno modification to tcp's fast recovery algorithm, April 2012. RFC6582.

[13] HUANG, J., QIAN, F., GUO, Y., ZHOU, Y., XU, Q., MAO, Z. M., SEN, S., AND SPATSCHECK, O. An in-depth study of LTE: effect of network protocol and application behavior on performance. In *ACM SIGCOMM 2013 Conference, SIGCOMM'13,*

Hong Kong, China, August 12-16, 2013 (2013), D. M. Chiu, J. Wang, P. Barford, and S. Seshan, Eds., ACM, pp. 363–374.

[14] JACOBSON, V. Congestion avoidance and control. In *Symposium Proceedings on Communications Architectures and Protocols* (New York, NY, USA, 1988), SIGCOMM '88, ACM, pp. 314–329.

[15] JAIN, R. *Art of Computer Systems Performance Analysis Techniques for Experimental Design Measurements Simulation and Modeling*. John Wiley & Sons, May 1991.

[16] JIANG, H., WANG, Y., LEE, K., AND RHEE, I. Tackling Bufferbloat in 3G/4G Networks. In *Proceedings of the ACM Conference on Internet Measurement Conference* (2012), IMC '12.

[17] KUNNIYUR, S., AND SRIKANT, R. Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management. SIGCOMM 2001.

[18] LUM TAN, W., LAM, F., AND CHEONG LAU, W. An Empirical Study on 3G Network Capacity and Performance. In *Proceedings of the 26th IEEE International Conference on Computer Communications. INFOCOM 2007.* (May 2007).

[19] MITTAL, R., SHERRY, J., RATNASAMY, R., AND SHENKER, S. Recursively Cautious Congestion Control. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)* (Seattle, WA, Apr. 2014), USENIX Association, pp. 373–385.

[20] NICHOLS, K., AND JACOBSON, V. Controlling Queue Delay. *Queue 10*, 5 (May 2012).

[21] NIKRAVESH, A., CHOFFNES, D. R., KATZ-BASSETT, E., MAO, Z. M., AND WELSH, M. Mobile Network Performance from User Devices: A Longitudinal, Multidimensional Analysis. In *Passive and Active Measurement - 15th International Conference, PAM 2014, Los Angeles, CA, USA, March 10-11, 2014, Proceedings* (2014), M. Faloutsos and A. Kuzmanovic, Eds., Springer, pp. 12–22.

[22] OPNET MODELER. http://www.opnet.com.

[23] RONG, P., PRABHAKAR, B., AND PSOUNIS, K. CHOKe - a stateless active queue management scheme for approximating fair bandwidth allocation. In *Proceedings of 19th Conference of the IEEE Computer and Communications Societies. INFOCOM 2000.* (2000), vol. 2.

[24] SCHOENEN, R., OTYAKMAZ, A., AND XU, Z. Resource Allocation and Scheduling in FDD Multihop Cellular Systems. In *ICC Workshops 2009. IEEE International Conference on* (June 2009), pp. 1–6.

[25] SHALUNOV, S. Low extra delay background transport. Internet-draft, Internet Engineering Task Force, 2010.

[26] SIVARAMAN, A., WINSTEIN, K., THAKER, P., AND BALAKRISHNAN, H. An Experimental Study of the Learnability of Congestion Control. In *ACM SIGCOMM 2014* (Chicago, IL, August 2014).

[27] VENKATARAMANI, A., KOKKU, R., AND DAHLIN, M. TCP Nice: A mechanism for background transfers. *ACM SIGOPS Operating Systems Review 36*, SI (2002).

[28] WINSTEIN, K., AND BALAKRISHNAN, H. End-to-end Transmission Control by Modeling Uncertainty About the Network State. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks* (New York, NY, USA, 2011), HotNets-X, ACM, pp. 19:1–19:6.

[29] WINSTEIN, K., AND BALAKRISHNAN, H. TCP Ex Machina: Computer-generated Congestion Control. In *Proceedings of the ACM SIGCOMM 2013 Conference* (2013).

[30] WINSTEIN, K., SIVARAMAN, A., AND BALAKRISHNAN, H. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation* (2013).

[31] XIA, Y., SUBRAMANIAN, L., STOICA, I., AND KALYANARAMAN, S. One More Bit is Enough. SIGCOMM 2005.

[32] ZAKI, Y., CHEN, J., PÖTSCH, T., AHMAD, T., AND SUBRAMANIAN, L. Dissecting Web Latency in Ghana. In *Proc. of the ACM Internet Measurement Conference (IMC)* (Vancouver, BC, Canada, Nov. 5-7 2014). Accepted.