Adaptive Congestion Control for Unpredictable Cellular Networks

Yasir Zaki
NYU Abu Dhabi
Abu Dhabi, UAE
yasir.zaki@nyu.edu

Thomas Pötsch
University of Bremen
Bremen, Germany
tpoetsch@uni-bremen.de

Jay Chen
NYU Abu Dhabi
Abu Dhabi, UAE
jchen@cs.nyu.edu

Lakshminarayanan Subramanian
NYU and CTED
New York, USA
lakshmi@cs.nyu.edu

Carmelita Görg
University of Bremen
Bremen, Germany
cg@comnets.uni-bremen.de

ABSTRACT

Legacy congestion controls including TCP and its variants are known to perform poorly over cellular networks due to highly variable capacities over short time scales, self-inflicted packet delays, and packet losses unrelated to congestion. To cope with these challenges, we present Verus, an end-to-end congestion control protocol that uses delay measurements to react quickly to the capacity changes in cellular networks without explicitly attempting to predict the cellular channel dynamics. The key idea of Verus is to continuously learn a delay profile that captures the relationship between end-to-end packet delay and outstanding window size over short epochs and uses this relationship to increment or decrement the window size based on the observed short-term packet delay variations. While the delay-based control is primarily for congestion avoidance, Verus uses standard TCP features including multiplicative decrease upon packet loss and slow start.

Through a combination of simulations, empirical evaluations using cellular network traces, and real-world evaluations against standard TCP flavors and state of the art protocols like Sprout, we show that Verus outperforms these protocols in cellular channels. In comparison to TCP Cubic, Verus achieves an order of magnitude (> 10x) reduction in delay over 3G and LTE networks while achieving comparable throughput (sometimes marginally higher). In comparison to Sprout, Verus achieves up to 30% higher throughput in rapidly changing cellular networks.

SIGCOMM '15, August 17 - 21, 2015, London, United Kingdom
 ISBN 978-1-4503-3542-3/15/08…$15.00
DOI: http://dx.doi.org/10.1145/2785956.2787498

CCS Concepts

•Networks → Network protocol design; Transport protocols; Network performance analysis;

Keywords

Congestion control, Cellular network, Transport protocol, Delay-based

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 [26, 20, 7]. TCP and its variants are known to perform poorly over cellular networks due to high capacity variability, self-inflicted queuing delays, stochastic packet losses that are not linked to congestion, and large bandwidth-delay products [15, 32, 33].

Three specific characteristics directly impact the unpredictability of cellular channels. First, the state of a cellular channel between a mobile device and a base station undergoes several complex state transitions that affect channel availability in short time scales [15]; this introduces variability in the underlying channel. Second, the frame scheduling algorithms used in cellular networks cause burstiness in the cellular channel. Based on real-world cellular measurements, we observe that the typical traffic characteristics at a receiver are bursty (even for smooth sending patterns) with variable burst sizes and burst inter-arrival periods. Third, while prior work has considered only self-inflicted queuing delay as a cause for high delays [33], we find that competing

traffic does affect end-to-end delay characteristics, especially under high contention or when the cellular channel is near saturation. Finally, device mobility has a substantial impact on channel characteristics that further compounds these challenges. The lack of channel predictability has important implications on the design of new congestion control protocols.

In this paper we present Verus, a delay-based congestion control protocol that is primarily designed for highly variable channel conditions that are hard to predict. Instead of attempting to predict the cellular channel dynamics, Verus uses cues from delay variations to track channel conditions and quickly change its sending window. The key idea of the Verus protocol is to remain in constant exploration mode and continuously learn a *delay profile* that captures the relationship between the sending window size and the perceived end-to-end delay. Using this delay profile and delay variation cues, Verus replaces the conventional Additive Increase (AI) in TCP with a series of increment/decrement steps to quickly adapt to changing channel conditions. While these control steps are primarily for congestion avoidance, Verus retains the loss-based multiplicative decrease step of TCP to quickly respond to congestion.

We implemented and tested Verus across a variety of environments, comparing it against Sprout and variants of TCP including Cubic, NewReno and Vegas. We then evaluated Verus using a combination of simulations, trace-based simulations, and real-world experiments. In these experiments we demonstrate that Verus achieves an interesting trade-off between the throughput and delay characteristics of Sprout and TCP variants in cellular channels. In comparison to TCP Cubic, Verus achieves an order of magnitude ($>$ 10x) reduction in delay over 3G and LTE networks while achieving comparable throughput (sometimes margin-ally higher). In comparison to Sprout, Verus achieves higher throughput under rapidly fluctuating channel conditions while maintaining low delay. Finally, we show that Verus provides good fairness properties when competing with other protocols and that it can rapidly adapt to highly variable channel conditions over short time scales.

2. RELATED WORK

Legacy Congestion Control Protocols

Congestion control is an extensively studied topic with numerous variants of TCP. TCP Reno [5], TCP Tahoe [16] and TCP NewReno [14] were among the early popular variants which are loss-based and TCP Vegas [3] was among the earliest delay-based control protocols. Most current operating systems leverage TCP Cubic [13] or Compound TCP [29]. While TCP Cubic makes specific modifications to the increment function in conventional AIMD-based window control, Compound TCP maintains two congestion windows to adapt its sending window. There are also a number of other TCP flavors such as LEDBAT [27], TCP Nice [30], equation based rate control [9], 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 [31]. 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 control drawing inspiration from protocols like TCP Vegas.

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. Examples of such techniques include Explicit Congestion Notification (ECN) [8], VCP [34], or active queue management like RED [10], BLUE [4], CHOKe [25], AVQ [19], CoDel [22]. 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

TCP has remained the gold standard for many years, but there have been several recent publications on new congestion control protocols for various environments. Sprout [33], for example, is a recent protocol 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 shows 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 later in this paper. Remy [32] focuses 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. [28] looked at the learnability of congestion control under imperfect knowledge of the network through an experimental study leveraging Remy as a design tool.

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) [21] shows that the initial small window of TCP during slow start often wastes several RTTs until the flow fully utilizes the available link bandwidth. RC3 uses several levels of lower priority services to achieve nearly optimal throughputs. Performance-oriented Congestion Control (PCC) proposes to empirically observe and adopt actions that result in high performance, but PCC's adaptation to "rapidly" changing

networks is on the order of seconds and does not consider unpredictable fluctuations on the order of milliseconds that occur in cellular networks [6].

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 "buffer-bloats" [12] and multi-second delays. Jiang et. al. [18] have shown the severity of bufferbloats through extensive measurements done on 3G and LTE commercial networks. The authors of [23] 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. Their results show that there are significant performance differences across carriers, access technologies, geographic regions and time. Zaki et. al. [35] conclude similar observations on cellular networks in developing regions as well as developed regions. Huang et. al. [15] studied the effect of network protocols on 3G and 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 (∼52%) under-utilize the available bandwidth of LTE.

3. CHANNEL UNPREDICTABILITY

The physical properties of radio propagation such as path-loss and slow-fading eventually cause changes in link performance despite mitigation techniques. As a result, cellular channels fluctuate rapidly over short time scales (milliseconds) and change more dramatically over slightly longer time scales (seconds). The inherent unpredictability of radio propagation along with the complex interactions between cellular networking components makes it difficult for simple channel prediction models to track channel variations and thus they motivate adaptive exploration protocols like Verus.

We present results from several experiments on commercial 3G/UMTS and LTE networks to highlight these issues. We make three specific observations:

- **Burst scheduling:** Typical traffic characteristics observed at a receiver are highly bursty with variable burst sizes and burst inter-arrival periods. Mobility further amplifies these characteristics.
- **Competing traffic:** When two or more flows contend for radio resources and their sending rates approach network capacity, we observe cross-flow dependencies.
- **Channel unpredictability:** Standard prediction mechanisms even using the most recent samples are far from capturing the bursty behavior of the channel.

We measured cellular network performance under several different conditions, investigating the effects of several factors including: data rates, mobility, competing traffic, and 3G/UMTS or LTE. The measurements were conducted on two commercial cellular networks, Du and Etisalat[1], for both downlink and uplink direction. Our measurement setup consisted of a standard rack server and a client laptop tethered to a Sony Xperia Z1 LTE mobile phone. We implemented a measurement tool that sends/receives UDP packets between the server and client at 0.4 ms sending intervals. We performed clock synchronization, tagged packets with sequence numbers, and included the sender timestamp to calculate the one-way delay at the receiver.

Burst Scheduling

Packet arrivals at receivers exhibit a cellular radio scheduler phenomenon 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 radio conditions that lead to sending a burst of several packets. Figure 1 illustrates this phenomenon for one of our LTE 10 Mbps downlink measurements.

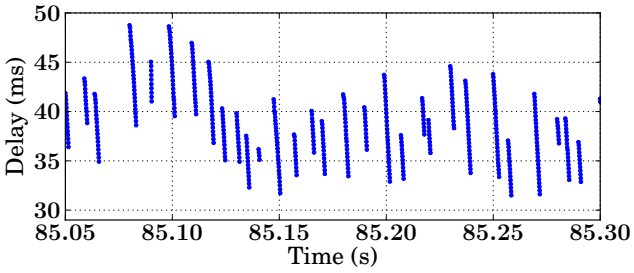


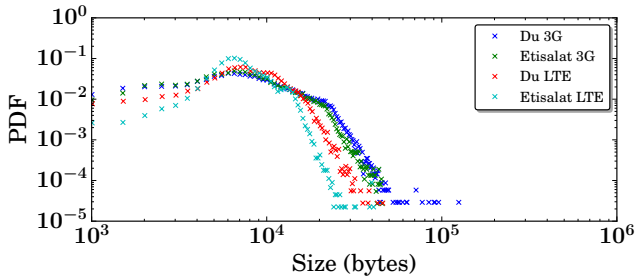
Figure 1: LTE 10 Mbps burst arrival time

Figure 2 summarizes our findings on burst scheduling for the two operators on 3G and LTE. In these downlink measurements our client was stationary and in an urban residential area. We observe that the burst size and inter-burst arrival time are difficult to predict and vary widely over the course of the 5 minute trace despite low contention and mobility. The LTE networks exhibit more frequent smaller bursts. Repeating this experiment while driving in the same area produces qualitatively similar results, but mobility causes both burst size and inter-arrival times to vary more widely. We make similar observations on the uplink.

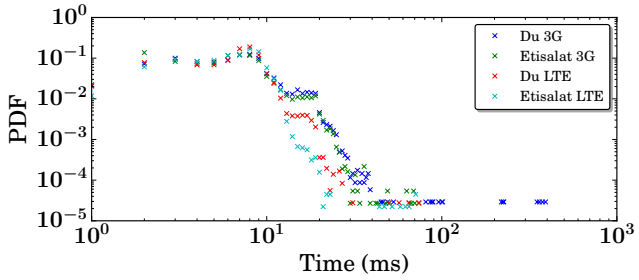
Competing Traffic

A common assumption made about cellular channels is that since the cellular scheduler maintains separate queues for each user, competing traffic flows may not affect each other. We perform a simple experiment to show that two competing flows especially close to link saturation can affect each other despite queue isolation,

[1]Etisalat and Du are two cellular providers in the United Arab Emirates (UAE) and they provide LTE coverage in most parts of the major cities.



(a) Burst size



(b) Burst inter-arrival time

Figure 2: Probability distributions 3G/LTE downlink

especially since these flows still compete for the same radio resources. We consider two users competing at the same cellular base station such that when both users are active, their combined data rates are almost equal to the 3G channel capacity. The first user is constantly receiving at a fixed rate (1, 5, 10 Mbps) while the second user is set to operate in ON/OFF periods of one minute intervals receiving at 10 Mbps. Figure 3 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 low, but when the second user is ON the average packet delay for the second user increases, especially when the combined data rate approaches the channel capacity.

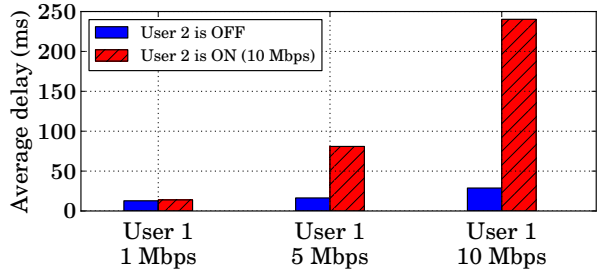
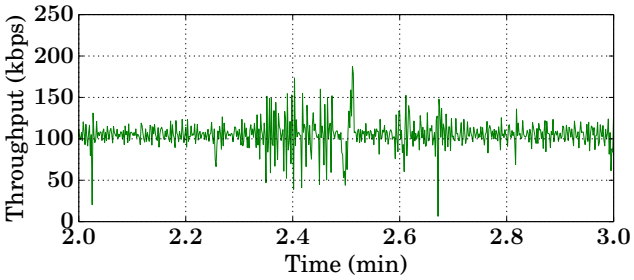


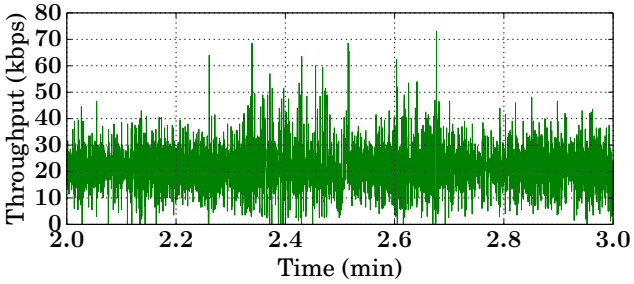
Figure 3: Impact of competing traffic on packet delay over a 3G downlink for User 1 when User 2 is ON/OFF

Channel Unpredictability

To demonstrate that cellular channels are non-trivial to predict, we used simple predictors to compare the measured data with the predicted data on a 3G downlink with one user receiving at 10 Mbps. Figure 4 is one representative trace from our campus parking lot on a 3G stationary downlink. We observe that even at 100 ms windows there are dramatic fluctuations in throughput due to burst scheduling. This variability is persistent and more evident at smaller timescales; if we zoom in to individual packet arrivals we observe the unpredictable variability previously illustrated in Figure 1 and Figure 2. To demonstrate that the channel is non-trivial to predict, we experimented with simple predictors to compare the predicted data with actual transmissions on 3G and LTE downlink. We found that linear predictors and k-step ahead predictors fail to track the high variations of the channel. While one could experiment with a variety of other predictors, the main result is that standard prediction mechanisms are far from capturing the bursty behavior of the channel despite the use of very recent samples.



(a) 100 ms windows



(b) 20 ms windows

Figure 4: Data received on a 3G stationary downlink at 100 ms and 20 ms window sizes

4. THE VERUS PROTOCOL

Verus is an end-to-end congestion control protocol designed for highly variable cellular channels that was heavily inspired by our channel observations. Since cellular channels are highly unpredictable, instead of attempting to predict the cellular channel dynamics, Verus uses delay variations to learn a *delay profile* that reflects the relationship between the network delay and the amount of data that can be sent without causing network congestion. Because contention and competing traffic impact performance, Verus takes into account delay feedback from the network to give an indication of contention and uses delay cues to constantly remain

in exploration mode rather than assume that delays are self-inflicted. Finally, because channel fluctuations occur at different time-scales, Verus uses small ε steps to track fast changes and delay profile updates to track slower changes.

Verus borrows a number of features from legacy TCP variants, such as slow start and multiplicative decrease, but changes the way it maintains the sending window. Legacy TCP uses additive increase and increases the congestion window (CWMD) size by 1/CWND, i.e. increasing the congestion window by one packet when it successfully received a full window. This process can be slow. In contrast, Verus increases/decreases the sending window at each ε ms epoch and adapts to the changing cellular channel by rapidly increasing the sending window when the channel conditions allows for more packets. Similarly, Verus seeks to reduce the sending window even before packet losses whereas TCP can only decrease the congestion window through an aggressive multiplicative decrease after a loss.

The main goal of Verus is to avoid congestion by maintaining an appropriate (sliding) sending window W over a period equal to the estimated network Round Trip Time (RTT). Verus does this by replacing the additive increase with a series of small ε steps to adapt quickly to channel fluctuations. Within a sending window, Verus estimates how many packets need to be sent to avoid congestion or packet loss over a smaller ε ms epoch. At each epoch (in the absence of losses) Verus either: increments or decrements W using the delay profile as follows:

$$W(t+1) = f(d(t) + \delta(t)) \tag{1}$$

where, $W(t+1)$ is the next sending window, f is the delay profile function with $d(t)$ being the network delay, and $\delta(t)$ is a delay increment or decrement value.

Verus builds a delay profile using the following four elements:

- **Delay Estimator**: estimates the network RTT using delay measurements reported from the receiver's acknowledgments
- **Delay Profiler**: tracks the relationship between delay and sending window that does not cause network congestion
- **Window Estimator**: estimates the sending window using the estimated delay and delay profile
- **Loss Handler**: handles losses and adjusts the sending window

Delay Estimator

The Delay Estimator is responsible for processing the receiver's acknowledgments and estimating the network delay (estimated network RTT). It calculates the packet round trip delay $D_{p,i}$ for each packet by subtracting the current time (i.e. ACK received time) from the packet sent time; where p represents the packet number and i represents the Verus epoch number.

The Delay Estimator keeps track of all received packet delays within a Verus epoch and stores them in a vector $\vec{D}_i$. This vector contains all received delay values $D_{p,i}$ during that epoch. In order to track the (short-term) channel history and to avoid abrupt changes, the maximum delay $D_{max,i}$ received within that epoch is weighted by an Exponential Weighted Moving Average (EWMA) and calculated as:

$$D_{max,i} = \alpha \cdot D_{max,i-1} + (1-\alpha) \cdot max(\vec{D}_i) \quad \text{with } 0 < \alpha \leq 1 \tag{2}$$

The difference between the averaged maximum delays of the last two epochs is denoted as ΔD_i and represents the increase/decrease of the maximum delay experienced by the network relative to the previous epoch:

$$\Delta D_i = D_{max,i} - D_{max,i-1} \tag{3}$$

ΔD_i is passed to the Window Estimator to estimate how much to send during the next epoch.

Delay Profiler

The creation and maintenance of the delay profile is used to estimate the next upcoming sending window W_{i+1} for epoch $i+1$. The delay profile can be represented as a graph (Figure 5) where the x and y axis correspond to the sending window (W_i) and the packet delay ($D_{p,i}$), respectively. In Verus, each packet p is sent as part of a sending window W_i, i.e. number of packets in flight. Verus keeps track of W_i for each sent packet and thus upon receiving an acknowledgement for packet p, the sender obtains a pair of $D_{p,i}$ and W_i and adds this data point to the delay profile.

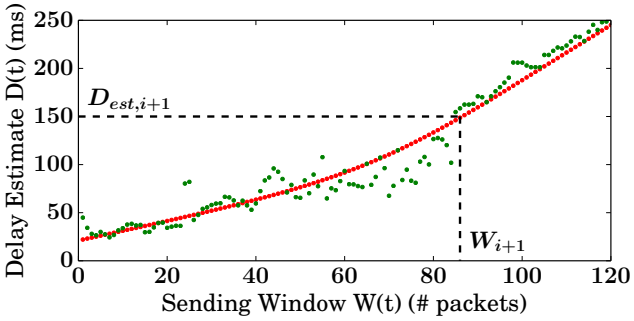


Figure 5: Example of Verus' delay profile. The green dots represent the recorded values and the red line reflects an interpolated delay profile curve.

Window Estimator

Verus uses the current change in the network delay ΔD_i to estimate the sending window W_i that is maintained over the current estimated network RTT. The sending window W_i is divided into smaller epochs with a fixed length to react quickly to channel changes. The next sending window estimate W_{i+1} is maintained over an average of the estimated network RTT. Figure 6 shows the basic time framework of Verus.

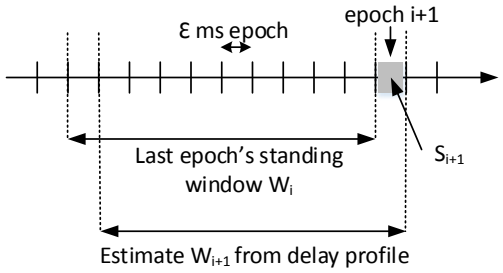


Figure 6: Verus time framework

Based on the sign of ΔD_i, Verus decides whether to increase or decrease the sending window. If ΔD_i is negative, it is an indication that the network/channel conditions are improving and thus more data can be sent to the network. If the ΔD_i is positive, the network/channel conditions may be experiencing congestion or negative changes and thus Verus should reduce the data rate. Verus estimates the delay that the network should have $D_{est,i+1}$ as follows:

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

where D_{min} is the minimum delay experienced by Verus, δ_1 and δ_2 are increment/decrement parameters, and R is the maximum tolerable ratio between D_{max} and D_{min}.[2] Verus then uses the delay estimate $D_{est,i+1}$ to find the corresponding sending window W_{i+1} on the delay profile (see Figure 5).

At the beginning of epoch $i+1$, Verus calculates the number of packets to send during this epoch. The number of packets to be sent within the next sending window W_{i+1} are calculated as follows:

$$S_{i+1} = max[0, (W_{i+1} + \frac{2-n}{n-1} \cdot W_i)] \quad \text{with } n = \lceil \frac{RTT}{\varepsilon} \rceil \quad (5)$$

where S_{i+1} is the number of packets to send during the epoch, W_{i+1} is the estimated sending window for the future, W_i is the current 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 per estimated network RTT.

Loss Handler

If Verus detects a packet loss or timeout, the sending window is reduced and the new W_{i+1} is multiplied by a multiplicative reduction factor as:

$$W_{i+1} = M \cdot W_{loss} \quad (6)$$

[2]R is used to tune the protocol trade-off between delay and throughput. We show in the evaluation section the effect of the value of R.

where, W_{loss} is the sending window in which the loss occurred, and M is the multiplicative decrease factor. We choose the sending window of the lost packet W_{loss} because that sending window was responsible for the packet loss.

Once a loss is identified and the sending window is multiplicatively decreased, Verus enters a loss recovery phase. During the loss recovery phase, the delay profile is no longer updated. The loss recovery phase is important because Verus builds its delay profile to reflect what could be sent without incurring network losses. Packets that arrive after a loss would have lower buffer delays and hence are not considered.

During the loss recovery phase and upon receiving an acknowledgement, the sending window W_{i+1} is increased by $1/(W_{i+1})$ (similar to TCP). Verus exits the loss recovery phase once acknowledgments of packets sent after the loss are received, i.e. if the protocol receives an acknowledgement with a sending window that is smaller than or equal to the current sending window. Verus also uses a timeout mechanism similar to TCP in case all packets are lost.

5. VERUS IMPLEMENTATION

Our prototype implementation of Verus consists of sender and receiver applications written in C++. The sender application runs in a multi-threaded environment and uses the real time extension library *librt*. As the underlying transport protocol, UDP is used to transmit the generated packets. A number of implementation details must be addressed in order to realize the Verus protocol in practice. These include delay profile initialization and maintenance, handling of timeouts and retransmissions, and setting of parameters.

5.1 Delay Profile Initialization and Maintenance

Verus relies heavily on the delay profile, which reflects the relationship between the network delay and the sending window without congesting the network. The initial creation of the delay profile is handled during Verus' slow start phase. Verus' slow start is similar to TCP's slow start; where the sender begins by sending a single packet towards the receiver and upon receiving an acknowledgement the sender increments the sending window by one, which leads to exponential growth of the sending window.

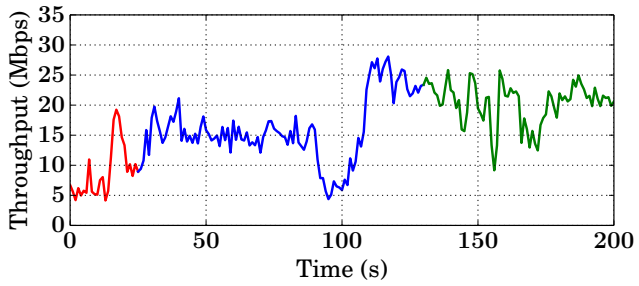
Verus maintains a list of sent packets and stores the sending timestamp as well as the sending window with which the packet was sent. The sender uses this information to calculate the packet RTT (i.e. delay) and records a (delay, sending window) tuple. Once one of the exit conditions for Verus slow start are met, the sender will have a number of delay/sending window tuples to build the delay profile. The delay profile is constructed from the stored tuples using the cubic spline interpolation from the *ALGLB* library.

Verus' slow start phase has two exit conditions:

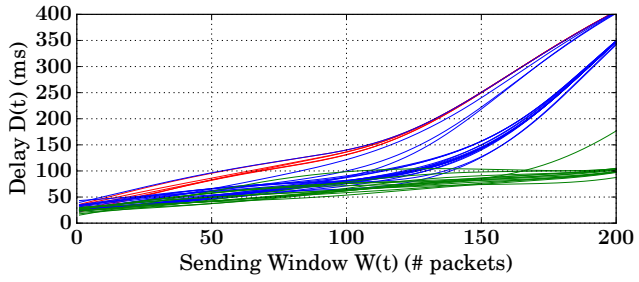
- encountering a packet loss: this can be deduced from acknowledgement sequence numbers
- the RTT delay exceeds the predefined threshold: this threshold is set as N x minimum delay (e.g., N=15)

During the course of operation, the delay profile needs to be updated and maintained over time to capture channel changes. The delay profile is updated as follows: for every received acknowledgement at the sender, the delay value of the point that corresponds to the sending window of the acknowledged packet is updated with the new RTT delay. This update is performed using an Exponentially Weighted Moving Averaging (EWMA) function to allow the delay profile to evolve. Due to the high computational effort of the cubic spline interpolation, this calculation is not performed after every acknowledgement, but instead at certain intervals. In Section 5.3 we discuss reasonable update intervals in more detail.

Figure 7b illustrates how the delay profile may evolve over time. For clarity, only every fifth interpolation is shown and we restricted the channel trace to 200 s. The three curves of each color correspond to the same colored region shown in the throughput graph in Figure 7a. It can be observed that the delay profile curve changes over time with respect to the fluctuations of the channel, i.e. the smaller the available throughput is, the steeper the delay profile becomes.



(a) Channel trace for downlink



(b) Verus delay profile evolution (excluding slow start)

Figure 7: Channel trace and the corresponding Verus delay profile curve evolution

5.2 Timeouts and Retransmissions

Although Verus is a congestion avoidance protocol designed to handle the fluctuating capacities of a cellular channel, packet losses are sometimes inevitable as an intrinsic property of the cellular medium. Our current implementation of Verus is built on top of UDP. Verus uses sequence numbers to keep track of received packets and their RTTs. These sequence numbers are used to identify packet losses at the sender. To deal with packet reordering, our implementation does the following: for every missing sequence number Verus creates a timeout timer of 3*delay. If the missing packet arrives before the timer expires, no packet loss is identified; otherwise, the sending window is multiplicatively decreased and the missing packet is retransmitted.

5.3 Verus Parameter Settings

Verus makes use of a variety of parameters and the selection of these parameters influences the performance of the protocol or substantially changes the overall protocol behavior. The effects of parameter changes are mainly reflected in throughput, delay, and fairness among flows. In our sensitivity analysis we wanted to identify the specific effects of parameter settings and to understand their relation to common scenarios.

Our sensitivity analysis of Verus parameters were performed using the OPNET network simulator. In order to emulate real cellular network behavior in OPNET, we collected channel traces in uplink and downlink direction from a commercial cellular network provider (Etisalat) and replayed these channel conditions in OPNET to schedule flows under contention.

The setup for collecting the traces consists of four Android smartphones (3x Samsung Galaxy S4 and 1x Sony Xperia Z1) and one server. As the server is connected via a fiber link directly to Etisalat's backbone network so that additional delays and unwanted background traffic are minimized. All smartphones are running a native sender and receiver application to communicate with a server located in our premises. The server runs the same sender and receiver application. Both endpoints, server and smartphones, send UDP packets with an MTU size of 1400 bytes simultaneously with a constant data rate to the other endpoint. The corresponding endpoint acts as a sink and records the timestamp of each packet arrival. We use this bi-directional setup to measure downlink and uplink of the channel.

As the measurement is executed on the 3G HSPA+ cellular network, the data rate for each device is set to 5 Mbps and 2.5 Mbps for downlink and uplink, respectively. These data rates are close to the upper limits of the network, but do not necessarily reflect the maximum capacity of the cellular network. The maximum capacity of the channel is difficult to determine and depends on many factors, e.g. cross-competing traffic, mobility, and interference. We expect that by using these data rates the channel is not over-saturated and packet buffering is minimized under ideal channel conditions.

In total, we ran the measurements for seven different scenarios to capture a variety of conditions with different mobility properties. Each measurement was conducted over five minutes and all devices were started at the same time and in the same location. The seven scenarios are the following: *Campus stationary, Campus pedestrian, City stationary, City driving, Highway driving, Shopping Mall and City waterfront.* The channel traces were generated from the packet arrival timetamps at the receiver and contain inter-arrival times between consecutive packet arrivals. Using these channel traces, our parameter sensitivity analysis in OPNET focused mainly on the following parameters: epoch time (ε), the delay profile update interval, and the delta decrement (δ_1) and delta increment (δ_2).

Epoch ε

In general, cellular channels have three different effects governing the changes, some are short-term (e.g., fast-fading) and others are more long-term (path-loss and slow-fading). The epoch ε determines in which intervals Verus calculates the amount of packets to send S_{i+1}. The smaller the epoch, the faster Verus reacts to fast-fading or other sudden channel changes. Through extensive simulation we found that an epoch of 5 ms is a good value. This value causes Verus to quickly adjust the operating point on the delay profile and adapt to sudden short-term fluctuations. Larger values of ε cause Verus to adjust the sending window too slowly to respond to such fluctuations. Within a 5 ms epoch, the cellular channel does not experience larger long-term channel changes caused by path-loss or slow-fading effects. Instead, these effects are handled by the delay profile update rate as described below.

Delay Profile Updates

Path-loss and slow-fading dramatically change channel conditions, and Verus must adapt the delay profile to match these new conditions. Feedback about channel conditions continuously update the data in the delay profile. Verus re-interpolates the delay profile at fixed time intervals. Our sensitivity analysis indicates that an update interval of 1 s shows reasonable results and is being used in this work. Re-interpolation intervals at higher than 1 s values start causing Verus to miss channel changes and react slowly to slow-fading channel changes. A much smaller update interval than 1 s is too aggressive since path-loss and slow-fading do not occur at such high frequency.

Delta δ_1 and δ_2

The deltas determine how restrictive (δ_1) or aggressive (δ_2) the protocol reacts to delay changes during each epoch. The larger the values, the stronger the effects. We find through simulation that their range should between 1 ms $\leq \delta \leq$ 2 ms with the condition that $\delta_1 \leq \delta_2$. Beyond these guidelines for good Verus performance, these parameters allow Verus behavior to be tuned to the level of desired fairness when competing with other protocols. For the evaluations in this paper we use values of 1 ms for δ_1 and 2 ms for δ_2.

6. VERUS MACRO-EVALUATION

In this section, we evaluate the two main macro-level properties of the performance of the Verus protocol, namely throughput and delay characteristics. We compare Verus against Sprout [33], TCP Cubic, TCP New Reno, and TCP Vegas. We take a two-pronged approach to evaluate Verus against these flavors of TCP and Sprout: real-world and trace-driven evaluation.

In the real-world evaluation, we use mobile devices (Samsung Galaxy S4 and Sony Experia Z1 phones) to run the protocols on 3G and LTE networks. In this evaluation, we are constrained by the number of devices that we are able to simultaneously use; we also restrict the number of flows per device.

In the trace-driven evaluation, we use multiple mobile devices simultaneously connecting to 3G or LTE networks where each device is coupled with traffic generators to generate realistic network traces of the channel under different conditions. Using the procedure described in 5.3, we specifically generate an additional set of traces that capture network contention and mobility scenarios. Then we use the OPNET network simulator to compare Verus with other TCP variants [24].[3]

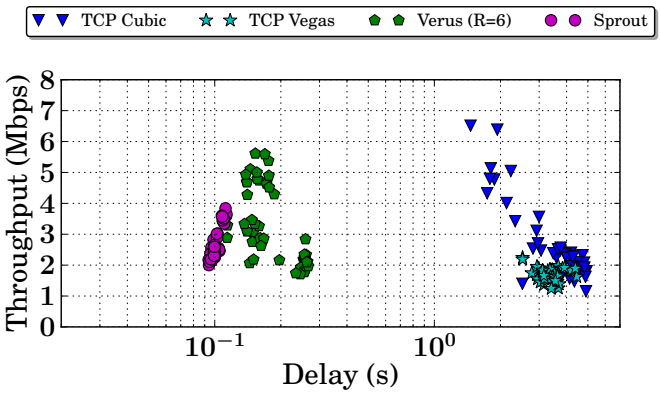
6.1 Real-world 3G and LTE Networks

We performed this evaluation on the Etisalat network, the largest cellular network operator in the UAE, which provides both 3G and LTE cellular network service. While Verus was running natively on the mobile devices, TCP Verus, TCP Cubic, and Sprout were executed on tethered laptops. We switched the mode of each phone to connect to the appropriate network to test both 3G and LTE. We consider multiple flows between the devices and a server with a public IP address at our university campus with high bandwidth and low network delays. To emulate contention, we simultaneously trigger competing flows between the different devices and the server. To avoid device contention issues, we limit the number of flows per device to three. Specifically, we consider the following scenarios:

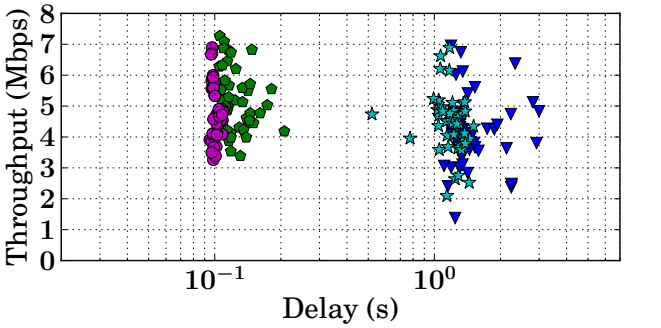
1. Three phones each running three Verus flows
2. Three phones each running three Sprout flows
3. Three phones each running three Cubic flows (secure copy (scp) download)
4. Three phones each running three Vegas flows (scp download)

All experiments were performed on Etisalat's 3G and LTE network at fixed locations without mobility and at the same time (late evening in a residential area). The duration of each run was two minutes and each experiment was repeated five times. All tests were done

[3]The Sprout codebase is not compatible with the OPNET simulator, thus we compare against Sprout only in the real-world evaluations. Furthermore, we use the "sendonly" implementation of Sprout to get a fair comparison to Verus and other protocols.

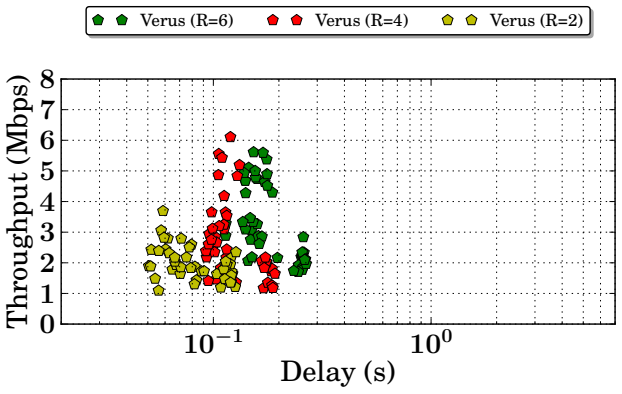


(a) 3G throughput vs. delay

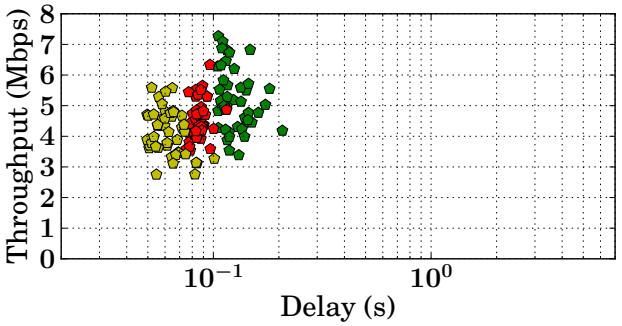


(b) LTE throughput vs. delay

Figure 8: Averaged throughput and delay of Sprout, TCP Cubic, TCP Vegas, and Verus on 3G and LTE



(a) 3G throughput vs. delay



(b) LTE throughput vs. delay

Figure 9: Different values of R in Verus trade-off higher throughput and lower delay

on the downlink and the results from each flow were averaged for each experiment.

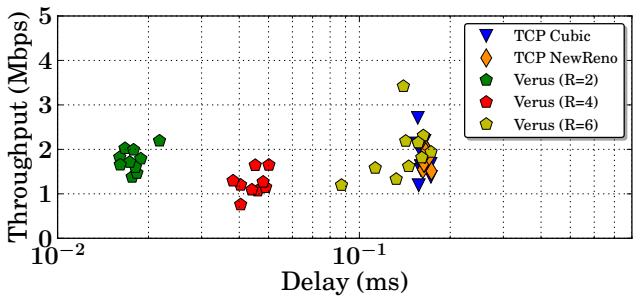
Figure 8 shows the average throughput and average delay for each of the flows across all runs on 3G and LTE. We make the following observations: The average delay observed by Verus flows is an order of magnitude lower than the average delay of TCP Cubic and TCP Vegas flows. Comparing 3G and LTE, Verus marginally outperforms TCP Cubic in terms of throughput in 3G network conditions and compares even more favorably in LTE networks. Also, in these scenarios Verus generally performs similar to Sprout with slightly higher throughput and higher delay.

For Verus we repeated the experiments for different values of the ratio R (the maximum tolerable ratio between D_{max} and D_{min}) to show how the value of R tunes Verus to different trade-offs between higher throughput and lower delay. Setting the Verus R parameter to six leads to throughputs higher than Sprout, but with slightly increased delays. Figure 9 shows the impact of the Verus R parameter on the protocol behavior. Depending on the value of R, the Verus protocol can be tuned to achieve a trade-off between a higher throughput or lower delay. By setting R to two, Verus achieves lower delay compared to Sprout with slightly lower throughput.

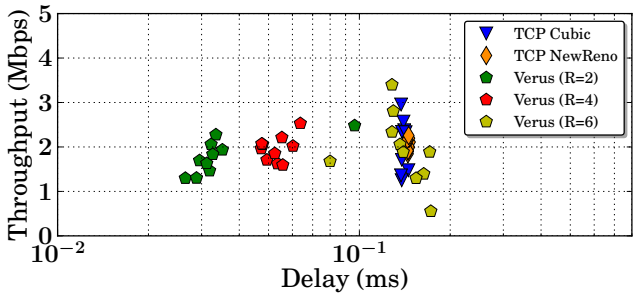
6.2 Trace-driven Evaluation

Traffic flows over cellular channels are not perfectly isolated and exhibit contention for radio resources. To evaluate Verus in a contention scenario with several competing flows, we rely on the OPNET simulator and use channel traces to emulate real cellular network behavior. Following the procedure in Section 5.3, we collected additional cellular channel traces to perform the evaluations in this section. The channel traces are fed into a traffic shaper and replayed upon packet arrival. In general, the traffic shaper is a modified version of a regular network router (as natively available in OPNET) and also implements a shared queue with Random Early Detection (RED) [11] queue management using the following parameters: minimum queue size 3 MBit, maximum queue size 9 MBit, and drop probability 10%.

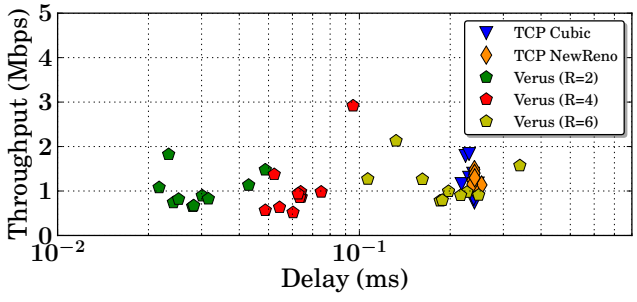
We evaluate Verus under high contention within the OPNET network simulator for each of these channel traces to understand how Verus performs under different competing traffic scenarios in terms of throughput and delay. Additionally, our simulations show fairness properties among the flows for the different protocols. We show the results primarily for the downlink direction but the observations are similar for the uplink. Our simulations are configured to run with 2, 5, 10 and 20



(a) Campus pedestrian



(b) Slow driving within the city with signals



(c) Fast driving on highway

Figure 10: Delay-throughput comparison with 10 flows for different mobility patterns

simultaneous clients for TCP Cubic, TCP NewReno, and Verus. Both TCP scenarios are configured with full buffer FTP traffic and default parameters according to Linux 3.16 (TCP Cubic) and Windows 7 (TCP NewReno) configurations. Verus is configured with the parameters obtained in Section 5.3.

Figure 10a shows a scatter plot of the throughput of all the individual flows of the average delay and average throughput observed by each of the flows in a campus pedestrian environment. Overall, we observe that Verus with lower R ratio experiences an order of magnitude lower delay than TCP Cubic or TCP New Reno while the throughput of Verus is comparable to the throughput of TCP Cubic and TCP New Reno. In all these settings, TCP Cubic and TCP New Reno are slow to adapt to varying channel conditions and therefore incur high buffering delays while Verus quickly adapts to channel variations and experiences much lower buffering delays. Increasing R to six, increases throughput, but also increases delay due to additional buffering.

Mobility

Figures 10b and 10c show the same scatter plot of Verus, TCP Cubic, and TCP New Reno flows under two other mobility patterns. We observe that while the average throughput remains roughly the same, mobility has an impact on the variance of the throughput across competing TCP flows. However, even with high mobility, the variation in throughput across Verus flows is small, which is indicative of Verus being able to quickly adapt and achieves high levels of fairness despite mobility.

Fairness

To better quantify the fairness argument, we consider Jain's fairness index [17] measured as:

$$f(x_1, x_2, ..., x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} {x_i}^2} \quad (7)$$

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 1 is perfect fairness. We compute Jain's fairness index over windows of one second and average these one second fairness values for the overall fairness for each protocol for comparison. Here, we vary the number of competing flows for each experiment between 2 to 20 flows. Here and in the following experiments, we set Verus' parameter $R = 2$ unless otherwise stated.

Scenario	TCP Cubic	TCP NewReno	Verus
2 Users	98.1%	89.7%	94.6%
5 Users	93.5%	86.3%	87.6%
10 Users	76.2%	83.8%	90.7%
15 Users	75.2%	83.3%	86.8%
20 Users	70.1%	82.0%	78.6%

Table 1: Jain's fairness index comparison

Table 1 shows Jain's fairness index for all three protocols. We report the average fairness index across all five different scenarios. Our results show that although TCP Cubic achieves high fairness among the scenarios with a small number of users, the fairness drops significantly under high contention (achieving about 70% fairness). In contrast, Verus shows slightly lower fairness compared to TCP Cubic for scenarios with a low number of users, while maintaining reasonable fairness at higher contention. TCP NewReno is almost consistent with the fairness it achieves across all scenarios, but achieves marginally lower fairness compared to Verus for scenarios with low number of users.

7. VERUS MICRO-EVALUATION

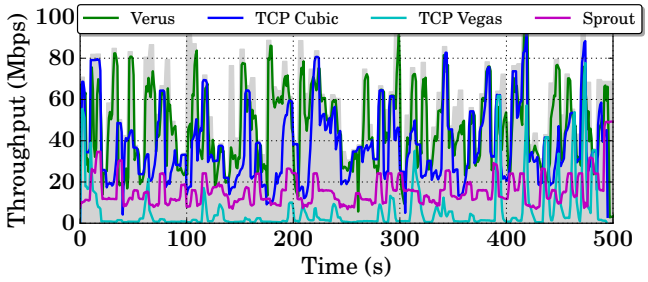
While the previous section detailed a macro-evaluation of Verus, in this section we focus on specific micro-evaluations of Verus to describe its fairness and adaptation properties. All the evaluations in this section are

performed in a simplified network configuration consisting of a dumbbell topology with three laptops connected to an Ethernet Gigabit switch, which in turn is connected to a server. The server outgoing bandwidth is controlled through the Linux Traffic Control (tc) tool. The tool is also used to emulate delays for each of the clients, this is used to configure the RTT.

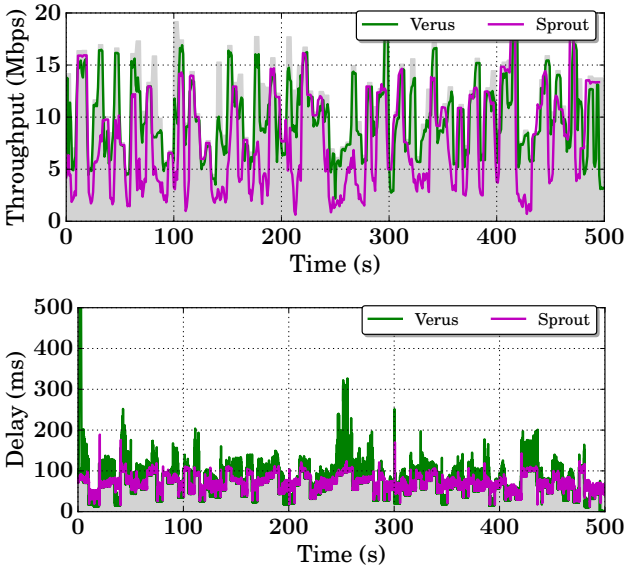
Rapidly Changing Networks

We wanted to measure how quickly Verus can adapt to high channel variations. This experiment is configured so that the network condition would change suddenly. Every five seconds the whole network parameters, i.e. link capacity, network RTT, and loss rate, are changed.

We have considered two variations of the experiment: where in the first version the network link capacity varied between 10 Mbps and 100 Mbps. whereas, in the second version, the link capacity varied between 2 Mbps and 20 Mbps. The reason behind the two versions was because the Sprout implementation bandwidth is capped at 18 Mbps. In both scenarios, the RTT was varied between 10 ms and 100 ms, and the loss rate between 0% and 1%.



(a) Scenario I



(b) Scenario II

Figure 11: Rapidly changing network evaluations under two different link capacities

Figure 11a shows the throughput over time for the first experiment where the gray shaded area represents the available link capacity. We observe that Verus outperforms the other protocols and manages to adapt very quickly to the rapid network changes. Understandably, Sprout does not perform well because of the bandwidth cap introduced by the Sprout implementation. Figure 11b shows the throughput and delay results of Verus compared to Sprout with a lower link capacity variation up to 20 Mbps. Here, we observe that Sprout performs better than before, but Verus still achieves higher throughput on average than Sprout.

Newly Arriving Flows

To understand the impact of the arrival of new flows, we consider a situation where eight competing Verus flows share a 90 Mbps bottleneck link. The experiment is configured so that every 30 seconds a new Verus flow starts, thus increasing the number of competing flows over time. Figure 12 shows the throughput of seven Verus clients over time. During the first 30 seconds when only one Verus flow is active, the flow is fully utilizing the 90 Mbps link. We observe that Verus quickly adapts to the arrival of new flows and also reaches good fairness across competing Verus flows when new flows arrive and depart.

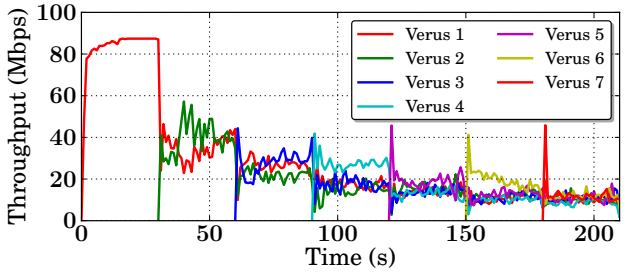


Figure 12: Verus intra-fairness among seven flows

Varying RTTs

To understand the impact of varying RTTs on Verus flows, we consider a simple experiment where three competing Verus flows with three different RTTs of 20 ms, 50 ms and 100 ms share a 60 Mbps bottleneck link. Figure 13 shows the temporal variation in the throughput of the different Verus flows. We observe that the throughput of Verus flows is independent of the RTT of the flows which is indicative that the Verus fairness model is close to Max-Min fairness. Given that delay-based protocols use non-linear control mechanisms, these protocols are harder to model analytically than conventional window-based protocols. We plan to develop a model to more fully characterize the behavior of Verus and other delay-based control protocols in future work.

Verus vs. TCP

One crucial issue when a new congestion control protocol is introduced is fairness to legacy TCP. We investigated how several Verus flows share an Ethernet bottleneck with several other TCP Cubic flows (since TCP

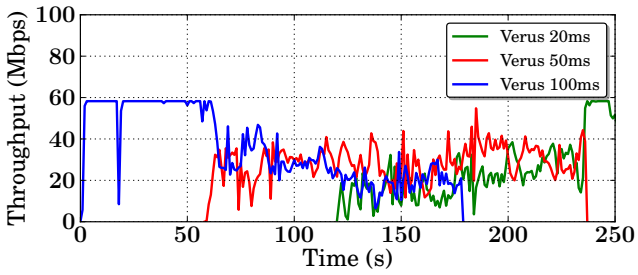


Figure 13: Verus intra-fairness with 3 different RTTs

Cubic is currently used as the standard in most TCP installations). The experiment consisted of three Verus and three TCP Cubic flows sharing a link capacity of 60 Mbps.

At the beginning of the experiment, every 30 seconds one new Verus flow is added to the network. Once all three Verus flows are present, a new TCP Cubic flow is added to the network every 30 seconds (at time 90, 120 and 150 seconds). The throughput comparison over time is shown in Figure 14. Our results show that Verus shares the bottleneck capacity equally with TCP Cubic.

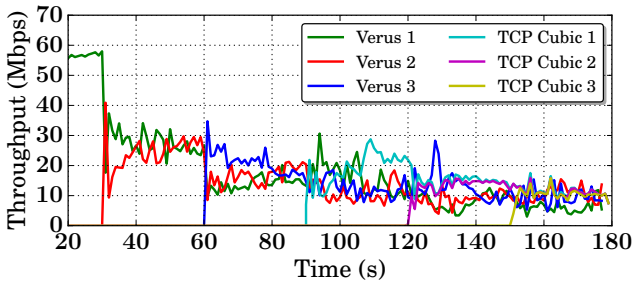


Figure 14: Verus fairness vs. TCP Cubic

Effect of Verus Delay Curve

In order to evaluate the effect of the delay profile curve, we compared two scenarios: one where the delay profile curve would update normally every second, and the second where Verus uses the first curve it generates without updating it. Figure 15 shows the results of this simulation for all of our five different collected traces. The results clearly show that updating the curve has an impact on performance due to the fact that the cellular channel changes and Verus needs to update its operating point on the curve based on these changes.

Short Flows

Short flows are a dominant feature in normal network traffic due to the nature of the commonly retrieved content. Although Verus is not specifically designed to cope with short flows, it is naturally able to handle short connections. When considering a short flow that does not progress beyond slow start, Verus behaves like legacy TCP due to the same slow start mechanism. After slow start, Verus uses the recorded delay profile to adapt quickly as it does with channel changes.

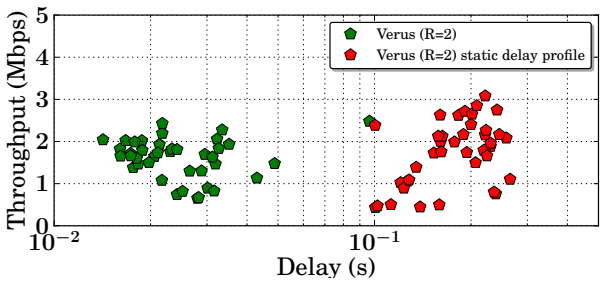


Figure 15: Verus with and without updating delay curve

8. CONCLUSIONS

In this paper, we presented Verus, an adaptive exploration congestion control protocol that is tailored for unpredictable cellular networks. Through continuous exploration using delay measurements and a delay profile, Verus adapts to both rapidly changing cellular conditions and to competing traffic. We tested Verus under a variety of experimental scenarios through simulation and real-world experiments over 3G and LTE networks. We show that in cellular networks Verus achieves higher throughput than TCP Cubic while maintaining a dramatically lower end-to-end delay, particularly over LTE. Verus also outperforms very recent congestion control protocols for cellular networks like Sprout under rapidly changing network conditions. In the future, we plan to experiment with other rapid adaptation mechanisms to theoretically characterize the behavior of Verus and develop a kernel implementation of Verus.

9. ACKNOWLEDGMENTS

We would like to thank Ali Raza for his help running measurements and experiments. We would also like to thank our shepherd Sachin Katti and the anonymous reviewers for their valuable feedback. We also want to extend our gratitude to Keith Winstein for his help with Sprout. Thomas Pötsch has been funded by the International Graduate School for Dynamics in Logistics, University of Bremen, Germany. We thank the the NYU Abu Dhabi Research institute and the Center for Technology and Economic Development (CTED) for supporting Lakshminarayanan Subramanian on this research work.

10. REFERENCES

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. DCTCP: Efficient Packet Transport for the Commoditized Data Center. *ACM SIGCOMM*, January 2010.

[2] D. Bansal and H. Balakrishnan. Binomial congestion control algorithms. In *Proceedings of 20th Conference of the IEEE Computer and Communications Societies. INFOCOM 2001*, Volume 2, 2001.

[3] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. *TCP Vegas: New techniques for congestion detection and avoidance*, volume 24. ACM, 1994.

[4] W. chang Feng, K. G. Shin, D. D. Kandlur, and D. Saha. The BLUE active queue management algorithms. *IEEE/ACM TRANS. NETWORKING*, pages 513–528, 2002.

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

[6] M. Dong, Q. Li, D. Zarchy, B. Godfrey, and M. Schapira. PCC: Re-architecting congestion control for consistent high performance. *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015.

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

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

[9] S. Floyd, M. Handley, J. Padhye, and J. Widmer. *Equation-based congestion control for unicast applications*, volume 30. ACM, 2000.

[10] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4), 1993.

[11] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

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

[13] S. Ha, I. Rhee, and L. Xu. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS Oper. Syst. Rev.*, 42(5), July 2008.

[14] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. The NewReno modification to TCP's fast recovery algorithm, April 2012. RFC6582.

[15] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, and O. Spatscheck. An in-depth study of LTE: effect of network protocol and application behavior on performance. In *ACM SIGCOMM 2013 Conference*, Hong Kong, China, August 12-16 2013.

[16] V. Jacobson. Congestion avoidance and control. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, pages 314–329, New York, NY, USA, 1988. ACM.

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

[18] H. Jiang, Y. Wang, K. Lee, and I. Rhee. Tackling Bufferbloat in 3G/4G Networks. In *Proceedings of the ACM Conference on Internet Measurement Conference (IMC)*, 2012.

[19] S. Kunniyur and R. Srikant. Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management. In *ACM SIGCOMM*, 2001.

[20] W. Lum Tan, F. Lam, and W. Cheong Lau. An Empirical Study on 3G Network Capacity and Performance. In *Proceedings of the 26th IEEE International Conference on Computer Communications. INFOCOM 2007.*, May 2007.

[21] R. Mittal, J. Sherry, R. Ratnasamy, and S. Shenker. Recursively Cautious Congestion Control. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 373–385, Seattle, WA, Apr. 2014. USENIX Association.

[22] K. Nichols and V. Jacobson. Controlling Queue Delay. *Queue*, 10(5), May 2012.

[23] A. Nikravesh, D. R. Choffnes, E. Katz-Bassett, Z. M. Mao, and M. Welsh. Mobile Network Performance from User Devices: A Longitudinal, Multidimensional Analysis. In M. Faloutsos and A. Kuzmanovic, editors, *Passive and Active Measurement - 15th International Conference, PAM 2014, Los Angeles, CA, USA, March 10-11, 2014, Proceedings*, pages 12–22. Springer, 2014.

[24] OPNET Modeler. http://www.opnet.com.

[25] P. Rong, B. Prabhakar, and K. Psounis. 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.*, volume 2, 2000.

[26] R. Schoenen, A. Otyakmaz, and Z. Xu. Resource Allocation and Scheduling in FDD Multihop Cellular Systems. In *ICC Workshops 2009. IEEE International Conference on*, pages 1–6, June 2009.

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

[28] A. Sivaraman, K. Winstein, P. Thaker, and H. Balakrishnan. An Experimental Study of the Learnability of Congestion Control. In *ACM SIGCOMM 2014*, Chicago, IL, August 2014.

[29] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A compound TCP approach for high-speed and long distance networks. In *Proceedings of 25th Conference of the IEEE Computer and Communications Societies. INFOCOM*, 2006.

[30] A. Venkataramani, R. Kokku, and M. Dahlin. TCP Nice: A mechanism for background transfers. *ACM SIGOPS Operating Systems Review*, 36(SI), 2002.

[31] K. Winstein and H. Balakrishnan. End-to-end Transmission Control by Modeling Uncertainty About the Network State. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, pages 19:1–19:6, New York, NY, USA, 2011. ACM.

[32] K. Winstein and H. Balakrishnan. TCP Ex Machina: Computer-generated Congestion Control. In *Proceedings of the ACM SIGCOMM 2013 Conference*, 2013.

[33] K. Winstein, A. Sivaraman, and H. Balakrishnan. 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.

[34] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman. One More Bit is Enough. SIGCOMM 2005, 2005.

[35] Y. Zaki, J. Chen, T. Pötsch, T. Ahmad, and L. Subramanian. Dissecting Web Latency in Ghana. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, Vancouver, BC, Canada, 2014.