URCP: Universal Rate Control Protocol for Real-Time Communication Applications

Di Xie [*][†], Sanjeev Mehrotra [*], Jin Li [*], Y. Charlie Hu [†]

[*] *Microsoft Research* – {dxie, sanjeevm, jinl}@microsoft.com

[†] *Purdue University* – ychu@purdue.edu

Abstract

We present a Universal Rate Control Protocol (URCP), which provides applications with *fair and full link utilization* while operating at *low delay and loss levels* as needed by interactive real-time communications (RTC) applications across a range of complex networks in use today, such as Wi-Fi hotspots, 4G (HSPA+), and WiMAX. As opposed to many existing rate control techniques which are specially designed for differing network types and application requirements, URCP attempts to provide a universal framework for rate control by learning network parameters and then automatically adapting a utility maximization based rate control framework to achieve good performance across any network.

URCP is able to provide significantly improved performance over state-of-art rate control techniques for RTC applications, in terms of throughput, delay, and loss. On real-world network tests, URCP provides similar throughput to commonly used loss based schemes while achieving up to a 100x improvement in operating delay (compared to TCP NewReno on the Clearwire WiMAX network). Compared to existing delay based rate control protocols commonly used by RTC applications, URCP provides similar operating delay while achieving close to 2.5x improvement in throughput on noisy networks (compared to WebRTC on the T-Mobile HSPA+ network).

I. Introduction

Networks are becoming increasingly more complex. In today's networks, we see an abundance of challenging network access links such as Wi-Fi hot spots, 3G/4G mobile broadband, and WiMAX. In addition, there are a significant number of middle boxes through which traffic is routed, for example gateway and proxy servers, network address translation (NAT) boxes, firewalls, intrusion detection devices, and traffic policers.

A common issue which occurs when traffic is routed over complex access links is the introduction of *inherent delay and loss noise* to packets traversing the network. For example, a device operating on wireless and cellular links (e.g., Wi-Fi hot spots, 3G/4G, and WiMAX) may incur additional packet delay and loss due to weak wireless signal or interference which leads to link layer retransmissions [11]. Scheduling policies of cellular networks and internet service providers (ISPs) (e.g. cable modem) may result in additional latency due to the device waiting for its turn to transmit packets [17], [27]. When a packet passes through network middleboxes, e.g., a 3G/4G gateway, a corporate firewall, or a network intrusion detection device, it may incur additional processing latency or loss if the CPU of the middlebox is overloaded or if there are software issues.

To be more precise, we define the observed network delay and packet loss that is *uncorrelated* with the transmission rate over the bottleneck link as the *inherent delay (δ) or loss (ϵ)*. This is in contrast to the *congestion-induced delay or loss* which is caused by *queuing* when the transmission rate is larger than the bottleneck link capacity. The *operating delay or loss* is the sum of the *inherent delay or loss* and the *congestion-induced delay or loss*. We can write

$$\delta_{operating} = \underbrace{\delta_{propagation} + \delta_{noise}}_{\delta_{inherent}} + \delta_{queuing}, \tag{1}$$

$$\begin{aligned}\epsilon_{operating} &= 1 - (1 - \epsilon_{noise})(1 - \epsilon_{queuing}) \\ &\approx \underbrace{\epsilon_{noise}}_{\epsilon_{inherent}} + \epsilon_{queuing}.\end{aligned} \tag{2}$$

That is the inherent delay noise, δ_{noise}, is defined to be the inherent delay observed on the network regardless of what transmission rate we are operating at minus the propagation delay (assumed to be the minimum delay seen). The inherent loss noise, ϵ_{noise} is defined to be the loss rate observed regardless of the transmission rate. The approximation in Eqn. 2 for $\epsilon_{operating}$ holds true for low loss rates.

Inherent delay and loss noise can present significant problems to rate control protocols which attempt to control the transmission rate for applications. Rate control techniques can be roughly classified into two main categories, (i) rate and window based congestion control techniques where the rate is controlled by increasing or decreasing the rate in response to congestion signals (delay, loss, and/or ECN) and (ii) available bandwidth estimation (ABE) techniques where the rate is directly estimated or forecasted as a function of congestion signals. Regardless of the exact technique being used, they all utilize congestion signals which are some combination of delay, loss, and/or ECN.

Since the *observed delay and loss* is the *operating delay and loss*, a typical way to estimate congestion signals caused by queuing is to use

$$\begin{aligned} \hat{\delta}_{queuing} &= \delta_{operating} - \delta_{propagation} \\ \hat{\epsilon}_{queuing} &= \epsilon_{operating}. \end{aligned} \tag{3}$$

Eqn. 3 is clearly a good estimate of congestion signals on clean networks such as data center networks, corporate infrastructure networks, and ISP core networks, where δ_{noise} and ϵ_{noise} in Eqn. 2 are close to zero. However, it does not hold true for noisy networks commonly seen today such as Wi-Fi hotspots, 3G/4G, and WiMAX.

For non-RTC applications, such as file download and video-on-demand (VOD), the primary measure that determines the performance is *long-term average throughput*. Since there are several rate control strategies that are able to achieve high throughput at the expense of high operating delay and loss levels, the performance of these applications does not suffer much on noisy networks. For example, a rate control strategy may only use loss as a congestion signal as opposed to delay since it is a less noisy signal. In addition, by allowing a larger queuing delay, such strategies can often prevent the mis-interpretation of noise for congestion [4] allowing for high link utilization as we will show in Sec. V.

The same strategies which are able to achieve full throughput for non-RTC applications cannot typically be used for RTC applications. We consider any interactive application that communicates among several parties (people or devices) with tight end-to-end (E2E) delay constraint as an *RTC application*. The performance of RTC applications depends critically on both the *short-term throughput* as well as the *instantaneous (per-packet)* delay and packet loss. For example, video conferencing has perceivable degradation to human observers if the E2E delay is larger than 200ms or the uncorrectable packet loss is larger than 1%. At the same time, these applications are also bandwidth intensive as a higher operating bandwidth leads to noticeable improvement in audio/visual quality. Therefore, strategies which deliver high link utilization at the expense of high operating delay and loss levels cannot be used. In addition, rate control strategies which operate at low delay and loss levels typically result in link underutilization over noisy networks, delivering sub-par performance.

Besides video conferencing, other examples of RTC applications include VoIP, virtual reality, online meetings (e.g. WebEx, Lync), tele-presence, online games, and interactive software applications (e.g. desktop remoting, application streaming, applications running in the cloud). With the rise of cloud computing, all categories of RTC applications are seeing rapid adoption and are extremely valuable.

In this paper, we propose to solve the problem of rate control for RTC applications on noisy networks while maintaining good performance on clean networks and for non-RTC applications as well. We develop a Universal Rate Control Protocol (URCP) which uses a utility maximization (UM) based rate control protocol which automatically adapts and adjusts itself to deliver fair and full link utilization while operating at low delay and loss levels across a range networks, including both clean and noisy networks. URCP is able to deliver as much throughput as protocols optimized for maximizing throughput but at delay and loss levels which are comparable to the inherent delay and loss present on the network. The additional delay and loss caused by congestion induced queuing is often insignificant when using URCP.

II. Noisy networks

We first study if inherent delay and loss noise is a real issue on networks commonly used to access the internet these days by first estimating the distribution of the delay and loss noise present on networks using the following.

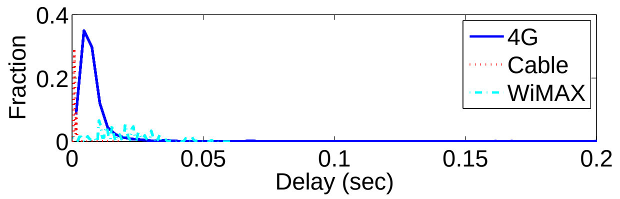
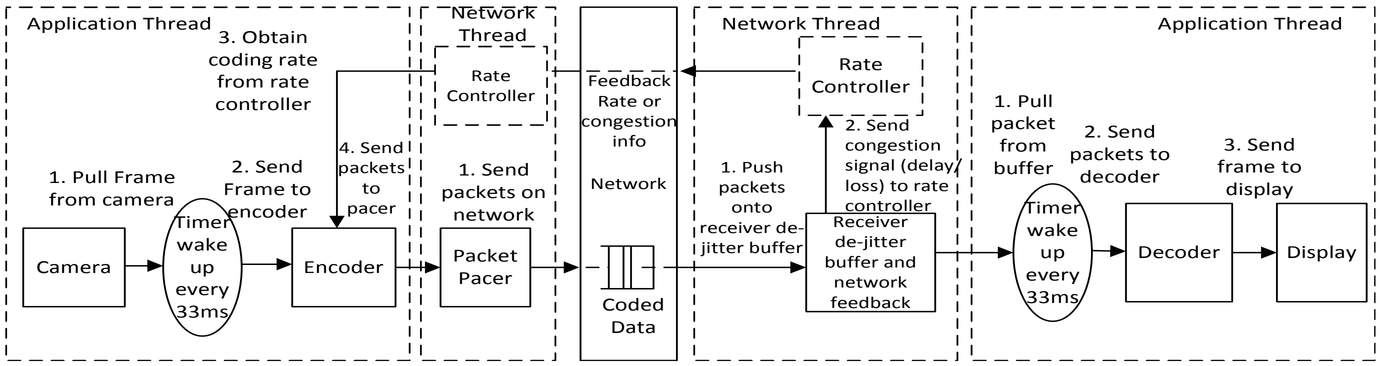


Fig. 1: Distribution of delay (estimate of δ_{noise}) over Cable modem, WiMAX, and 4G (HSPA+) networks when sending at 50% of estimated link capacity.



Application Thread
3. Obtain coding rate from rate controller
Network Thread
Rate Controller
Feedback Rate or congestion info
Network
Network Thread
Rate Controller
2. Send congestion signal (delay/ loss) to rate controller
Application Thread
1. Pull Frame from camera
2. Send Frame to encoder
4. Send packets to pacer
1. Send packets on network
1. Push packets onto receiver de-jitter buffer
1. Pull packet from buffer
2. Send packets to decoder
3. Send frame to display
Camera
Timer wake up every 33ms
Encoder
Packet Pacer
Coded Data
Receiver de-jitter buffer and network feedback
Timer wake up every 33ms
Decoder
Display

Fig. 2: Video conferencing application showing a sender and receiver component.

We use one-way delay (OWD) as the delay measure, δ. In practice, some algorithms may use round-trip time RTT as the delay measure. We also approximate the propagation delay $\delta_{propagation}$ as the minimum OWD observed and subtract out the propagation delay from the observed (operating) delay to obtain the delay measure δ, i.e.,

$$\begin{aligned} \delta = OWD - OWD_{min} &= \delta_{operating} - \delta_{propagation} \\ &= \delta_{queuing} + \delta_{noise}. \end{aligned} \tag{4}$$

The loss measure, ϵ, is assumed to directly be the observed loss rate, $\epsilon \approx \epsilon_{queuing} + \epsilon_{noise}$.

To perform the experiment, we first send a 1 MB file over the network using TCP and measure the receiving rate. We then send data over UDP at a constant bitrate which is 50% of this measured rate and compute the delay measure, δ using Eqn. 4 for every packet. Since we are sending at a relatively low rate (at most 50% of capacity), δ and ϵ are good estimates for δ_{noise} and ϵ_{noise}.

We plot the distribution of δ in Fig. 1 for the Cable modem, WiMAX, and 4G (HSPA+) networks. We see that the Cable modem network has a very narrow distribution with $\delta_{noise} \approx 0$. However, for the WiMAX and 4G networks, this is not the case, with the average inherent noise being in 30-50ms range. The actual maximum delay noise for the 4G network is actually close to 350ms, but in order to better visualize the results, we have truncated the plot to 200ms. From this, we conclude that the inherent delay noise is highly variable depending on the network and that it is significant enough on certain networks to affect protocols using delay as a congestion signal.

The loss noise, ϵ_{noise}, for each of the three networks is 0.0012 for the cable modem, 0.0037 for the 4G (HSPA+), and 0.0185 for the WiMAX network. The noise for the WiMAX network is significant and in this case could affect loss based rate control protocols as well.

III. Rate Control in RTC Applications

In this section, we give an overview of the typical dataflow used by a video conferencing application, shown in Fig. 2. Other RTC applications will have a similar dataflow with the common element being that the E2E delay between content generation and content consumption needs to be low (on the order of network latencies).

At the sender side, the application thread is a loop with a timer which wakes up every $\frac{1}{fps}$ seconds, where fps is the frame rate in seconds. At that time, a frame is captured from the camera and encoded using a certain number of bits depending on the rate controller's specified bitrate and the fps. The encoded frame is partitioned into packets (if more than one packet is needed) and sent to a packet pacer. The packet pacer runs on the network thread and sends packets over the network to enforce the rate controller's sending rate and smooth out potential burst of packets in traffic. This is important for HD video conferencing to avoid a large burst of packets that may cause additional delay noise.

At the receiver side, the network thread places received packets into a de-jitter buffer, which is needed since packets may experience varying delay over the network, either due to the inherent delay noise or due to congestion-induced queuing delay. This thread also computes congestion signals such as packet delay and packet loss which are fed to the rate controller. The application thread wakes up every $\frac{1}{fps}$ seconds, retrieves packets from the de-jitter buffer, and sends them to the decoder which generates frames of audio or video to render.

The rate controller may reside at either the sender side or receiver side. If it is at the receiver side, sender timestamps in the packet header and the receive time can be used to obtain one-way delay (OWD) estimates using clock compensation [23]. Loss can be computed using sequence numbers present in the packet header. OWD and loss information can be used to compute a new sending rate which is then sent to the sender. If the rate controller resides at the sender side, acknowledgments containing sequence numbers and receive timestamps of the packets being acknowledged can be used to compute OWD and loss information which can be used to adjust the sending rate. The sending rate and/or congestion information feedback is piggybacked onto coded data to reduce overhead.

We define the end-to-end (E2E) delay as the time difference between when the frame is captured from the camera to the time it is rendered at the receiver. It can be broken into the following components

$$\delta_{E2E} = \delta_{encoder} + \delta_{pacer} + \delta_{network} + \delta_{de-jitter} + \delta_{decoder}. \tag{5}$$

ITU-T Recommendation G. 114 states that VoIP users will not notice delay if E2E delay is kept under 150ms. For video conferencing applications, an additional delay of 50ms (total E2E delay of 200ms) is tolerable. It recommends a limit of 400ms E2E delay for network planning purposes.

The encoder delay, $\delta_{encoder}$, and decoder delay, $\delta_{decoder}$, are determined by the computation capability of the sending and receiving device. The pacer delay δ_{pacer} is determined primarily by the traffic pattern of the RTC application. $\delta_{network}$ is the *operating delay* of the network rate control protocol, i.e. it is the $\delta_{operating}$ in Eqn. 2. Since the *inherent network delay* cannot be adjusted, the rate control protocol for an RTC application attempts to maximize link utilization while minimizing the congestion induced queuing delay $\delta_{queuing}$. The rate controller indirectly also affects the de-jitter buffer delay $\delta_{de-jitter}$ which needs to be set to a sufficient size in order to absorb network delay variation. Congestion-induced loss $\epsilon_{queuing}$ also needs to be minimized. by the rate control protocol.

IV. State-of-Art Rate Control Protocols and Related Work

We first review existing state-of-art rate control protocols and study their feasibility for use with RTC applications on noisy networks.

A. Available Bandwidth Estimation

Available bandwidth estimation (ABE) techniques [13], [19], [25], [28], [29] use the *packet delay variation (PDV)* to estimate the available bandwidth. PDV is defined as the difference in end-to-end OWD between a pair of packets with any lost packets being ignored. ABE techniques are commonly used by many existing video conferencing and other RTC applications. The reason for their popularity is that they can quickly converge on a rate (i.e., the available bandwidth estimate) through a few packet pairs. Another advantage is that the algorithm does not need to calculate clock-skew between the sender and the receiver. For a pair of packets, its PDV can be calculated as the receive time spread (often referred to as the inter-arrival time) minus send time spread. If the packet pair is sent back-to-back, the send time spread is zero, and only the inter-arrival time needs to be measured.

PDV estimates are used to either directly estimate available bandwidth [28] or fed through either a Kalman filter [19] or a probabilistic inference algorithm [29] which estimates the bandwidth using standard queuing models for the network.

Several limitations of ABE based rate control techniques are discussed in [14], with fairness across multiple flows being one of the major ones. In addition, ABE techniques often fail to deliver link utilization across noisy networks without manually tuning parameters in the network model for different networks. For example, [29] uses a network model specifically tuned for a cellular network.

B. TCP Congestion Control

TCP is a well studied protocol, with many variants of congestion control in the literature. TCP uses a congestion window W to control the number of bytes that can be outstanding at any time over the network. For a given round trip time RTT, the average sending rate is roughly $R \approx \frac{W}{RTT}$. TCP variants typically use delay and/or loss as congestion signals to adjust the congestion window, W.

Loss-based TCP congestion control techniques include TCP New-Reno [7], [12] and CUBIC TCP [9]. In TCP New-Reno, the TCP sender additively increases (AI) its window every acknowledgment by α until it sees a loss at which point it multiplicatively decreases (MD) the window by β, often referred to as AIMD behavior [7], [12].

Although loss-based TCP congestion control techniques often achieve close to full link utilization across many networks, this occurs at the expense of high operating delay levels as shown in Sec. VII. This is because on networks with low inherent loss noise, loss occurs only when buffers are full. If the buffer size is large, congestion induced loss implies a high congestion induced delay. Thus they are not suitable for use with RTC applications.

Delay-based TCP variants such as TCP Vegas [3], [26] have been proposed to deal with high queuing delays present with loss-based congestion control schemes. In TCP Vegas, the protocol increases or decreases the congestion window W by α by estimating the bottleneck queue depth using delay. Specifically, if $\frac{W\delta}{RTT} < \gamma_1$, the window is increased by α; and if $\frac{W\delta}{RTT} > \gamma_2$, the window is decreased by α. Upon loss, TCP Vegas multiplicatively decreases the window by β.

For clean networks, such as networks in data center, delay-based TCP congestion control can deliver good throughput at low operating delay levels. However, for noisy networks, it tends to under-utilize the network bandwidth. In addition, due to the additive-increase, additive-decrease (AIAD) property in its delay adaptation stage, TCP Vegas has issues in fairly sharing bandwidth with other flows [26].

C. TFRC

TFRC (TCP-friendly Rate Control) has also been proposed for media applications. In TFRC [10], a sending rate is determined by looking at congestion signals and then determining the sending rate by using the throughput equation for that protocol. However, if TFRC is using the throughput equation derived from a loss based TCP protocol, then the resulting operating delay and packet loss will be identical as that of TCP rate control, with the exception that the sending rate will be smoother.

D. Delay-Based Utility Maximization

Since the seminal framework introduced by Kelly, [16] and Low [18], network utility maximization (UM) has attracted significant attention in the research community. In this framework, network rate control protocols are considered as distributed algorithms that maximize aggregate user utility subject to network resource constraints. It provides a powerful tool to reverse engineer existing TCP protocol designs [15] and also allows systematic design of new protocols, see [6] for a detailed discussion.

Utility maximization (UM) based rate control attempts to maximize the total network utility, often using the *log-utility function* $U(R) = k_0 \log(R)$, subject to deterministic worst delay guarantees. After applying the primal-dual algorithm via a fluid limit analysis, the rate control equation in utility maximization takes the form:

$$\begin{aligned} \Delta R &= k_2(k_0 - \rho R(t))\Delta T \\ R(t + \Delta T) &= \max(R(t) + \Delta R, 0), \end{aligned} \tag{6}$$

where $R(t)$ is the rate at time t, ρ is an estimate of the congestion level, ΔT is the time since the last update, and k_0 and k_2 are parameters in the protocol. Typically ρ is assumed to be the delay measure from Eqn. 4.

Although delay-based UM schemes have good operating delay performance across all networks and good throughput on clean networks, they fail to saturate the link on noisy networks.

E. Other Related Work

TCP congestion control protocols using ECN have also been proposed to reduce queue lengths such as Data Center TCP (DCTCP) [1]. Although ECN markings should be relatively clean and not subject to noise, ECN markings are not widely used in the Internet today and thus DCTCP is well suited to data center environments where there is greater control on network configuration.

The issue of adapting to differing network conditions has also been discussed in the literature. In AdaVegas [20], the authors adapt parameters in TCP Vegas to improve the convergence rate. However, they do not discuss the issue of adapting to noisy networks. The work of Generalized AIMD [31] also considers variable parameters to allow a protocol to be *TCP fair*. However, this only shows the relationship between α and β which allows for the same operating point as TCP NewReno, but there is no mention of how to set α and β for a given network. In [2], the authors discuss how one can dynamically switch between two pairs of (α, β) to give a smoother rate when needed, but does not address the issue of adapting for noisy network conditions. In [22], manual parameter tuning is proposed to deal with the issue of noise on mobile networks. However, there is no automatic adaptation strategy presented.

To the best of our knowledge, existing work does not propose to automatically adapt the rate control protocol in response to varying network conditions as we are proposing in URCP.

V. Protocol Performance

Before explaining the design of URCP, we first examine the fundamental reason why existing protocols fail to deliver satisfactory performance for RTC applications on noisy networks and examine what is needed to design a protocol which can work well.

A. Protocol Operating Point

With the exception of ABE techniques, each of the rate control protocols presented in Sec. IV has an *operating congestion signal point* which is either the *operating delay* or the *operating loss rate*. This *operating point* can be written as a function of (i) the steady-state rate, (ii) the operating rate R, (iii) parameters in the rate control protocol, and (iv) optionally the RTT. In addition, each of the protocols has parameters which control the *rate of convergence* and this is independent of the *operating point*.

For example, in the TCP NewReno case, the Mathis equation [21] is widely accepted as stating the relationship between throughput, delay, and *congestion induced packet loss* and is given by $R = \frac{0.93\alpha}{RTT\sqrt{\epsilon}}$, where R is the steady-state rate, α is the maximum packet size, ϵ is the loss rate, and RTT is the round-trip time. From this we can write the *operating loss rate* for TCP NewReno as $\epsilon = \left(\frac{0.93\alpha}{RTT*R}\right)^2$.

For TCP Vegas, if we are operating in the delay regime, the *operating delay point* is given by $\delta = \frac{\gamma}{R}$, where γ is the TCP Vegas parameter that controls how many packets are to be queued within the network.

For UM based rate control presented in Sec. IV-D, the *operating delay point* is $\delta = \frac{k_0}{R}$, at which point the rate adjustment ΔR is zero. The k_2 parameter controls the rate of convergence but does not affect the operating delay point.

B. Impact of Inherent Delay and Loss Noise

We see that all rate control protocols have a *decreasing operating congestion signal point* (delay or loss level) with *increasing operating rate, R*. That is we can write,

$$\begin{aligned} \rho_{operating} &= f(R_{operating}, \theta) \\ &= \rho_{noise} + \rho_{queuing} \end{aligned} \tag{7}$$

where $\rho_{operating}$ is the operating congestion signal point (either operating delay measure or loss measure), $R_{operating}$ is the final operating rate, θ are protocol parameters, and f is a function which *decreases with increasing $R_{operating}$*. In Eqn. 7, we assume that the entire *inherent* measure of the congestion signal is caused by noise (for example, by removing propagation delay to obtain the delay measure as in Eqn. 4).

Since, the portion of congestion signal from queuing caused by the protocol, $\rho_{queuing}$, cannot be negative, $\rho_{operating} \geq \rho_{noise}$. Since f decreases with increasing $R_{operating}$ and since it is a function of θ, we can make the following to conclusions regarding protocol performance.

1) For a fixed value of θ, the maximum achievable operating rate $R_{operating}$ decreases with increasing congestion signal noise ρ_{noise}.
2) For a given congestion signal noise, ρ_{noise}, we can find protocol parameters θ that allow for an arbitrarily large operating rate $R_{operating}$ (up to the link capacity). That is we can find θ so that

$$\rho_{operating}(R_{operating}, \theta) \geq \rho_{noise}. \tag{8}$$

This forms the basis for an adaptive parameter design which allows for a protocol with high link utilization at low operating congestion levels.

C. Achieving Full Link Utilization

Eqn. 8 becomes a necessary and sufficient condition for full link utilization. Since the noise is a stochastic random variable, we can write

$$\delta_{operating}(R_{operating}, \theta) \geq E[\delta_{noise}] \tag{9}$$

$$\epsilon_{operating}(R_{operating}, \theta) \geq E[\epsilon_{noise}], \tag{10}$$

for the operating delay and loss points, where $E[.]$ is the expectation or average value of the noise.

Let $\bar{R}^{MAX}$ be the true bottleneck link capacity. Since by definition $R_{operating} \leq \bar{R}^{MAX}$, in order for a protocol to achieve full link utilization, it is sufficient (but not necessary) for the operating congestion level when operating at the bottleneck link capacity to be larger than or equal to the inherent congestion level noise present in the network. That is $\rho_{operating}(\bar{R}^{MAX}, \theta) \geq \rho_{noise}$ is a sufficient condition.

VI. URCP

Since the delay-based UM protocol presented in Sec. IV-D provides nice fairness and convergence properties in addition to good performance in the presence of clean networks [5], URCP starts with Eqn. 4 as the fundamental rate change equation. However, from Eqn. 8, we see that other protocols can also utilize similar strategies to achieve the desired properties.

From Eqn. 6, for the delay-based UM rate control protocol, the operating point is controlled by k_0 and is given by

$$\delta_{operating} = \frac{k_0}{R}. \tag{11}$$

Combining this with Eqn. 10 gives the following conditions for full link utilization

$$k_0 \geq R_{operating} E[\delta_{noise}], \qquad \text{(necessary and sufficient)} \tag{12}$$

$$k_0 \geq \bar{R}^{MAX} E[\delta_{noise}], \qquad \text{(sufficient).} \tag{13}$$

Although $k_0 = R_{operating} E[\delta_{noise}]$ is the minimum value we can use, it is not really practical as it requires prior knowledge of the operating rate which is what the rate controller is supposed to find.

However, directly using the sufficient condition also has issues in that over-estimations in $\bar{R}^{MAX}$ as well as multiple flows cause a linear increase in k_0 which causes a linear increase in operating delay which is bad for RTC applications. For example, if $\bar{R}^{MAX}$ is over-estimated by a factor of N or if N flows are sharing the link, then k_0 will be N times larger than the minimum resulting in the operating delay being N times larger than $E[\delta_{noise}]$, with queuing delay being $(N-1)E[\delta_{noise}]$.

To alleviate these issues, we would like to use Eqn. 12. Since we cannot directly use $R_{operating}$, an alternative is to use $k_0 = R^{AVE} E[\delta_{noise}]$, where R^{AVE} is a *sliding window average* of the operating rate. By using a different timescale, we can allow for both R and δ to converge.

TABLE I: The three stages of adaptation in URCP.

At time t, when a new packet is received, add current rate $R(t)$ to sliding window used to compute R^{AVE} and add current packet delay measure δ to sliding window used to compute δ^{AVE}. Update bottleneck link capacity estimate using $R^{MAX} \leftarrow \max(R^{MAX}, R^{AVE}). \quad (14)$ Update inherent delay noise estimate using $\delta^{MIN} \leftarrow \min(\delta^{MIN}, \delta^{AVE}). \quad (15)$
Update k_0 using $k_0 = \left(\sqrt{\frac{R^{MAX}}{R^{AVE}}} \delta^{MIN} + \beta(\delta^{AVE} - \delta^{MIN}) \right) R^{AVE}. \quad (16)$ Update k_2 using $k_2 = \frac{R^{MAX}}{k_0 M \sqrt{\frac{R}{R^{MAX}}} RTT_{min}}, \quad (17)$
Update rate R using $\Delta R = k_2(k_0 - \delta R(t))\Delta T$ $R(t + \Delta T) = \max(R(t) + \Delta R, 0), \quad (18)$

A. Protocol Adaptation

We use the notation $\bar{R}^{MAX}$ to be the actual *bottleneck link capacity* and define the term $\bar{\delta}^{MIN} = E[\delta_{noise}]$ to be the *average inherent delay noise* on the network. We use the terms R^{MAX} and δ^{MIN} to be their corresponding estimates. We also define the term R^{AVE} to be the average rate (as specified by the rate controller) over a sliding window of certain duration and δ^{AVE} to be the average observed delay measure $(OWD - OWD_{min})$ over the sliding window.

As opposed to existing rate control protocols which simply adapt the rate, URCP performs the following three-stage adaptation shown in Table I, (i) updating estimates for R^{MAX} and δ^{MIN}, (ii) updating the parameters k_0 and k_2, and (iii) updating the actual rate, R, each using successively smaller timescales. β and M are parameters in URCP which we will discuss in Secs. VI-B and VI-C.

The initial value of δ^{MIN} (estimate of average delay noise) is obtained by first sending data at a low rate and averaging the delay measure for the corresponding packets. The initial value of R^{MAX} (estimate of capacity) is obtained by using a slow start (exponential) ramp-up. Other methods could also be used to obtain these. The initial value of k_0 is set using Eqn. 13.

B. k_0 Adaptation

k_0 adapts according to Eqn. 16 where β is a parameter between 0 and 1. The term $\sqrt{\frac{R^{MAX}}{R^{AVE}}}\delta^{MIN} + \beta(\delta^{AVE} - \delta^{MIN})$ in k_0 is the *target operating delay*. If we simply set the operating delay to δ^{MIN}, then the operating point would be independent of rate and fairness could not be obtained, since two flows of differing rates could both have $\Delta R = 0$. Using the term $\sqrt{\frac{R^{MAX}}{R^{AVE}}}\delta^{MIN}$ allows for the flow with smaller rate to have larger k_0 and thus a higher operating delay, allowing it to take rate from the flow with a larger share. However it also ensures that the operating delay changes as a square root of the $\frac{R^{MAX}}{R^{AVE}}$ which is a desirable property for RTC applications. For example, N flows would only increase this term in the operating delay by $\sqrt{N}$.

The term $\beta\delta^{AVE}$ is present in order to allow URCP to effectively compete with more aggressive (TCP-style) flows as well as deal with under-estimations of δ^{MIN}. A TCP flow will increase δ^{AVE} and thus URCP will increase its k_0 in response to this. However, by using $\beta < 1$, there is a downward pressure to bring the operating delay back to $\frac{(\sqrt{\frac{R^{MAX}}{R^{AVE}}}-\beta)}{1-\beta}\delta^{MIN}$.

1) Convergence and operating point: Since in steady state $R \approx R^{AVE}$ and $\delta \approx \delta^{AVE}$, if we solve for the case when $\Delta R = 0$ in Eqn. 18, we get the following delay/rate operating point ($\delta_{operating}, R_{operating}$),

$$\delta_{operating} = \max\left(\bar{\delta}^{MIN}, \delta^{MIN}\frac{\sqrt{\frac{R^{MAX}}{R_{operating}}}-\beta}{1-\beta}\right),$$
$$R_{operating} = \min\left(\bar{R}^{MAX}, \frac{R^{MAX}}{\left(\frac{\delta_{operating}}{\delta^{MIN}}(1-\beta)+\beta\right)^2}\right). \tag{19}$$

If δ^{MIN} and R^{MAX} are accurate estimates of $\bar{\delta}^{MIN}$ and $\bar{R}^{MAX}$ respectively, then in the single flow case, we see that the operating point is $\delta_{operating} \approx \delta^{MIN}$ and $R \approx R^{MAX}$, that is we get close to full link utilization with delay levels that are close to the inherent delay noise on the network link (i.e. no noticeable queuing delay will be present).

Since the delay-based UM scheme with fixed k_0 has nice convergence properties, URCP also *maintains the same convergence properties* provided the estimates of R^{MAX} and δ^{MIN} converge. By definition, R^{MAX} and δ^{MIN} will have to converge and cannot oscillate since the the all time maximum can never decrease (Eqn. 14) from the previous value and the minimum can never increase (Eqn. 15). Thus convergence is guaranteed in URCP.

2) Effect of β: In URCP, the parameter β is similar to k_0 in the delay-based UM rate control with fixed parameters since it affects the operating congestion point. In the single flow case, if δ^{MIN} and R^{MAX} are accurate estimates, from Eqn. 19, we see that $\delta_{operating} \approx \delta^{MIN}$ and $R \approx R^{MAX}$ regardless of β. Thus, no parameter tuning is needed to get optimal performance in the single-flow case.

However, if either δ^{MIN} or R^{MAX} are inaccurate or if we have multiple flows, β affects the operating point. In particular, if $\beta = 0$, then overestimates in R^{MAX} and multiple flows cause the operating delay point to only increase as $\sqrt{\frac{R^{MAX}}{R_{operating}}}$ which is desirable for keeping the operating delay low. For example, 10 flows will only increase this term in the operating point by a factor of $\sqrt{10}$ instead of 10. If we underestimate δ^{MIN}, then the achievable rate decreases as $\left(\frac{\bar{\delta}^{MIN}}{\delta^{MIN}}\right)^2$. For example, if we underestimate δ^{MIN} by a factor of two, then the achievable rate is only $\frac{1}{4}$ of capacity. Under-estimates in R^{MAX} and over-estimates in δ^{MIN} are easily corrected and not an issue as can be seen from Eqns. 14 and 15.

On the other hand if $\beta = 1.0$, then underestimating δ^{MIN} does not cause any reduction in the achievable rate. However, this comes at the expense of the operating delay growing without bound. For example, even if R^{MAX} is slightly larger than $R_{operating}$, we see infinite delay from Eqn. 19.

The trade-off between increasing operating point with increasing $\frac{R^{MAX}}{R_{operating}}$ and decreasing achievable rate with increasing $\frac{\bar{\delta}^{MIN}}{\delta_{operating}}$ is unavoidable. Through experimentation, we set $\beta = 0.8$ in our implementation.

In the fixed parameter delay-based UM protocol, the operating delay point and the achievable rate are functions of k_0, R^{MAX} and δ^{MIN}. However, in URCP, from Eqn. 19, we see that the operating delay point and the achievable rate are functions of β, $\frac{R^{MAX}}{R_{operating}}$, and $\frac{\delta_{operating}}{\delta^{MIN}}$. From this we conclude that the choice of the parameter β is *independent* of the absolute value of R^{MAX} and δ^{MIN}, but rather is a function of the ratios $\frac{R^{MAX}}{R_{operating}}$ and $\frac{\delta_{operating}}{\delta^{MIN}}$. This allows us to choose β which is completely *independent* of the network, thus making URCP *universal*.

C. k_2 Adaptation

The parameter k_2 in the delay-based UM scheme affects the rate of convergence and has no effect on the operating point. Although k_2 can be fixed while k_0 adapts, we also adapt k_2 using Eqn. 17, where M is a parameter which controls the rate of convergence and the amount of oscillation in steady-state. Since $\Delta R = k_2 k_0$ if $\delta = 0$, we can view the term $M\sqrt{\frac{R}{R^{MAX}}}$ as the number of RTT_{min} until we reach a rate of R^{MAX}. The factor $\sqrt{\frac{R}{R^{MAX}}}$ makes the number of RTT_{min} small when R is small relative to R^{MAX}. As R approaches R^{MAX}, the number of RTT_{min} approaches M.

For example if $M = 100$, then when $R = \frac{R^{MAX}}{10}$, then, we try to reach R^{MAX} in $\frac{100}{\sqrt{10}} = 31.6$ round-trip times. This allows us to ramp up faster when we are far from R^{MAX} and as R approaches R^{MAX}, we adaptively lower k_2. After experimentation on several networks, we set $M = 100$.

The rate of convergence parameter M is also independent of the network since it is scaled by $\sqrt{\frac{R}{R^{MAX}}}$ and reflects the number of RTT_{min} needed to reach R^{MAX}.

D. Fairness

Since the fixed parameter delay-based UM protocol is fair with respect to multiple flows if all flows have the same k_0, *URCP will also be fair* provided all flows converge to the same value of k_0. From Eqn. 19, we see that this will happen provided all flows have the same estimates of the bottleneck link capacity R^{MAX} and the inherent delay noise δ^{MIN}. Since these are relatively stable network parameters which can be estimated, URCP is fair with respect to multiple flows. This fact is validated empirically in Sec. VII.

Fairness can also be guaranteed if for example there was some central server which knew the network topology and characteristics which could notify each flow with accurate values of R^{MAX} and δ^{MIN} (or directly k_0 and k_2), ensuring that each flow uses the same k_0. In addition, if we repeatedly connect from the same endpoint (say the same 4G connection or cable connection), then we can use pre-cached values to initialize these parameters thus obtaining more accurate and stable estimates.

E. Multiple Congestion Signals

URCP can be easily modified to incorporate multiple congestion signals such as delay, loss, and ECN, by using multiple URCP rate controllers each reacting to different congestion signals using Table I and each specifying a rate (e.g. R^{δ} from controller reacting to delay and R^{ϵ} from controller reacting to loss). Then, we can define the actual rate R to be the minimum of all rates, e.g. $R = \min(R^{\delta}, R^{\epsilon})$. The actual operating congestion level then becomes the minimum of the operating congestion levels from each controller.

For example, in the shallow buffer case, if $\delta_{operating}$ is larger than the buffer size, the operating delay is not achievable and R^{δ} will grow without bound. However, R^{ϵ} will be controlled from the controller which reacts to loss and we will achieve an operating loss rate of $\epsilon_{operating}$.

VII. Evaluation

We use several real-world networks to show the performance of URCP: ADSL, cable modem, Ethernet (with rate limiting), public Wi-Fi hotspot, WiMAX (Clearwire), and 4G (HSPA+) (T-Mobile). We use a server connected to a well provisioned network on one end of the connection. At the other end, we use a client connected to each of the networks mentioned above. In our sessions, the client sends data to the server. In this setup, the client's connection is the bottleneck link. If we reverse the direction of traffic, we get similar results as those presented here. In our implementation, we implement the rate control on the sender side in Fig. 2.

In the evaluations, we measure the following network parameters: throughput (receiving rate), operating delay measure, and operating packet loss. The *operating delay measure* is computed using Eqn. 4 by subtracting the minimum observed delay from the observed delay. The *operating packet loss* is simply the observed packet loss rate. We note that the operating delay measure presented is the sum of both the *inherent delay noise* and the *congestion-induced queuing delay* from Eqn. 4.

We test URCP and compare with the following state-of-art rate control protocols in the literature.

1) TFRC: We use the TFRC throughput equation based on TCP NewReno from [10] in our test application.
2) TCP NewReno congestion control: We use the default implementation in Windows and use `tcpdump` to obtain the performance an RTC application would see.
3) TCP Vegas: We use the implementation provided in Linux [24] and port the congestion control portion to our test application.
4) WebRTC: We use the rate control specified in [19].

TABLE II: Comparison of URCP, TFRC using TCP NewReno throughput equation, Loss-based TCP New Reno (labeled TCP-NR), Delay-based TCP Vegas (labeled TCP-V), WebRTC, and delay-based UM (labeled UM) rate control protocols over various networks showing throughput (in kbps), operating delay (in msec), loss rate (as a fraction), and PSNR of video sequence (in dB).

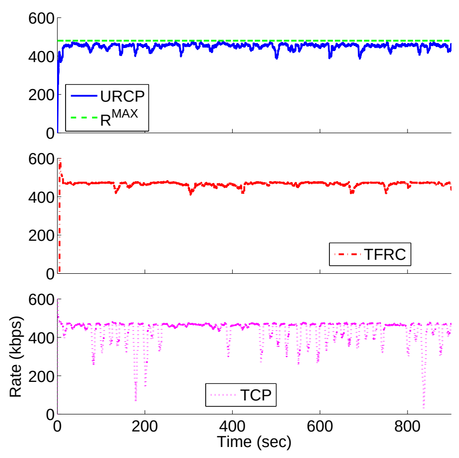
Network	Prot.	Avg. Rate (kbps)	Avg. Delay (msec)	90% delay (msec)	Loss Rate	Avg. PSNR (dB)
ADSL	URCP	753.3	11	16	0.0022	47.3
	TFRC	820.5	442	724	0.0004	36.3
	TCP-NR	825.3	860	1295	0.0057	32.4
	TCP-V	661.4	23	24	0.0070	44.8
	WebRTC	636.7	19	32	0.0003	46.5
	UM	733.6	20	33	0.0043	46.6
Cable Modem	URCP	990.6	14	27	0.0005	48.3
	TFRC	986.6	562	830	0.0002	29.8
	TCP-NR	1001.8	386	489	0.0003	30.2
	TCP-V	814.9	13	22	0.0027	46.7
	WebRTC	292.5	2	3	0.1219	28.6
	UM	1039.2	13	23	0.0002	48.5
Ethernet	URCP	8136.0	0.77	1.10	0.0000	52.4
	TFRC	3865.9	2.51	5.74	0.0542	46.5
	TCP-NR	9218.1	6.90	6.98	0.0250	48.3
	TCP-V	6834.1	5.36	10.4	0.0001	52.4
	WebRTC	7989.5	5.27	12.0	0.0000	52.3
	UM	6871.1	2.07	3.78	0.0000	52.3
Wi-Fi	URCP	516.4	41	82	0.0249	44.7
	TFRC	515.6	136	214	0.0296	38.7
	TCP-NR	501.7	252	403	0.0164	34.0
	TCP-V	485.6	78	195	0.0244	40.5
	WebRTC	461.0	16	42	0.0020	44.4
	UM	486.2	35	109	0.0099	43.3
WiMAX	URCP	451.1	26	41	0.0002	45.1
	TFRC	467.4	526	1010	0.0007	30.8
	TCP-NR	459.0	2050	2926	0.0010	28.9
	TCP-V	378.6	60	53	0.0020	42.0
	WebRTC	212.2	58	94	0.0000	40.5
	UM	332.6	57	105	0.0080	41.5
4G (HSPA+)	URCP	1491.9	37	69	0.0039	50.2
	TFRC	1306.2	100	185	0.0179	44.2
	TCP-NR	1472.7	281	401	0.0026	42.8
	TCP-V	665.1	27	36	0.0178	45.3
	WebRTC	639.7	39	53	0.0008	47.5
	UM	639.2	25	32	0.0683	44.5

5) Delay-based UM using fixed k_0 and k_2: We implement the rate control using Eqn. 6 with $k_0 = 16000$ bits and $k_2 = 0.6$.

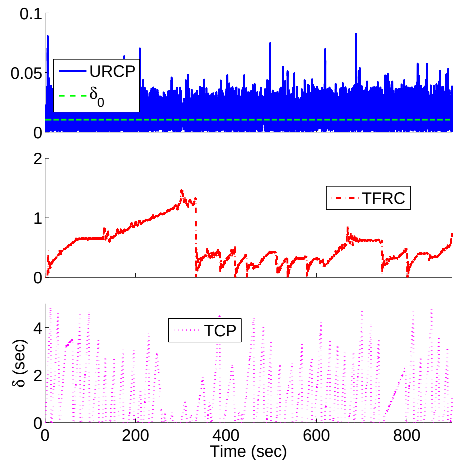
In addition to just comparing network metrics, we also estimate the video quality we would get if the rate control protocol was used in a video conferencing application. The video is encoded using the JM reference software encoder [8] using a fast encoding mode with a single reference picture and is representative of what would be used in a video conferencing setting. Using the packet level trace generated by each protocol when running over a real network, we determine whether the packet is received or lost and it's reception time. The video is then decoded using the JM reference software using the error concealment provided. The decoder has a fixed size dejitter buffer and packets which arrive with a delay larger than it's size are also considered lost. The PSNR between the original and reconstructed frames is computed for the luminance (Y) component.

TABLE III: Results obtained when initial parameters, R^{MAX} and δ^{MIN} are mis-estimated.

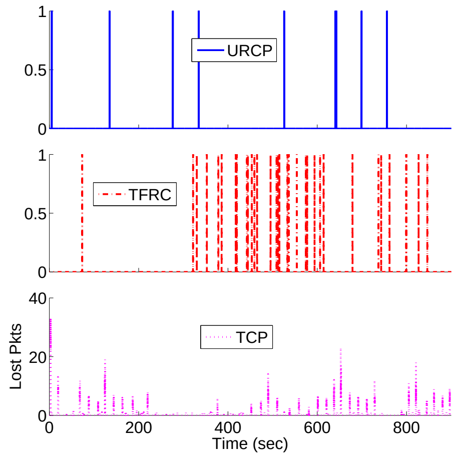
Network	Param	Avg. Rate (kbps)	Avg. Delay (msec)	90% delay (msec)	Loss Rate	Init R^{MAX} (kbps)	Final R^{MAX} (kbps)	Init δ^{MIN} (msec)	Final δ^{MIN} (msec)
ADSL	Correct	753.30	11	16	0.0022	841.90	841.90	5	4
	$R^{MAX} = 2\bar{R}^{MAX}$	762.97	21	28	0.0178	1683.79	1683.79	5	2
	$R^{MAX} = .5\bar{R}^{MAX}$	728.92	8	11	0.0010	420.95	822.96	5	5
	$\delta^{MIN} = 2\bar{\delta}^{MIN}$	772.04	60	121	0.0075	841.90	841.90	10	8
	$\delta^{MIN} = .5\bar{\delta}^{MIN}$	598.13	8	11	0.0009	841.90	841.90	3	2
Wi-Fi	Correct	512.23	45	92	0.0199	555.21	591.67	50	16
	$R^{MAX} = 2\bar{R}^{MAX}$	558.62	194	309	0.0294	1110.42	1110.42	50	47
	$R^{MAX} = .5\bar{R}^{MAX}$	513.56	32	69	0.0078	277.61	600.24	50	8
	$\delta^{MIN} = 2\bar{\delta}^{MIN}$	520.62	90	172	0.0390	555.21	597.52	100	49
	$\delta^{MIN} = .5\bar{\delta}^{MIN}$	458.80	76	166	0.0191	555.21	583.93	25	25
Ethernet	Correct	8136.05	0.77	1.10	0.0000	9086.86	9086.86	1	0.59
	$R^{MAX} = 2\bar{R}^{MAX}$	8490.08	1.79	2.35	0.0002	18173.73	18173.73	1	0.54
	$R^{MAX} = .5\bar{R}^{MAX}$	7771.81	0.69	0.82	0.0002	4543.43	8589.81	1	0.50
	$\delta^{MIN} = 2\bar{\delta}^{MIN}$	8566.19	0.82	0.97	0.0001	9086.86	9086.86	2	0.73
	$\delta^{MIN} = .5\bar{\delta}^{MIN}$	6073.68	0.69	0.76	0.0001	9086.86	9086.86	0.5	0.25



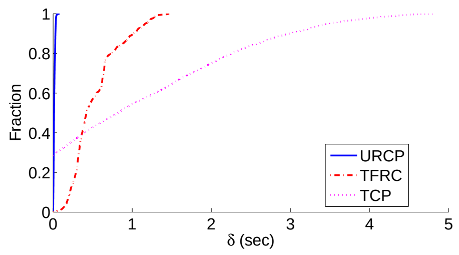
(a) Transmission Rate



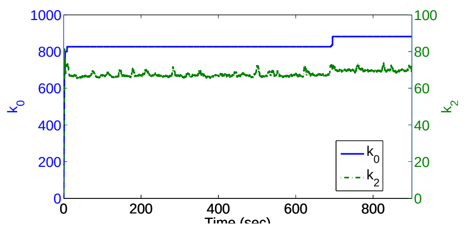
(b) Operating Delay



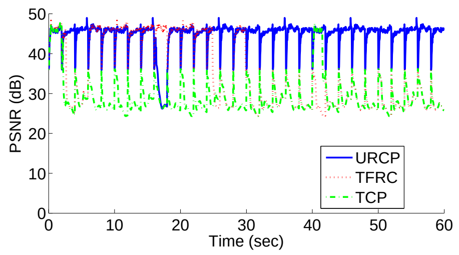
(c) Packet Loss



(d) CDF of Operating Delay



(e) k_0, k_2

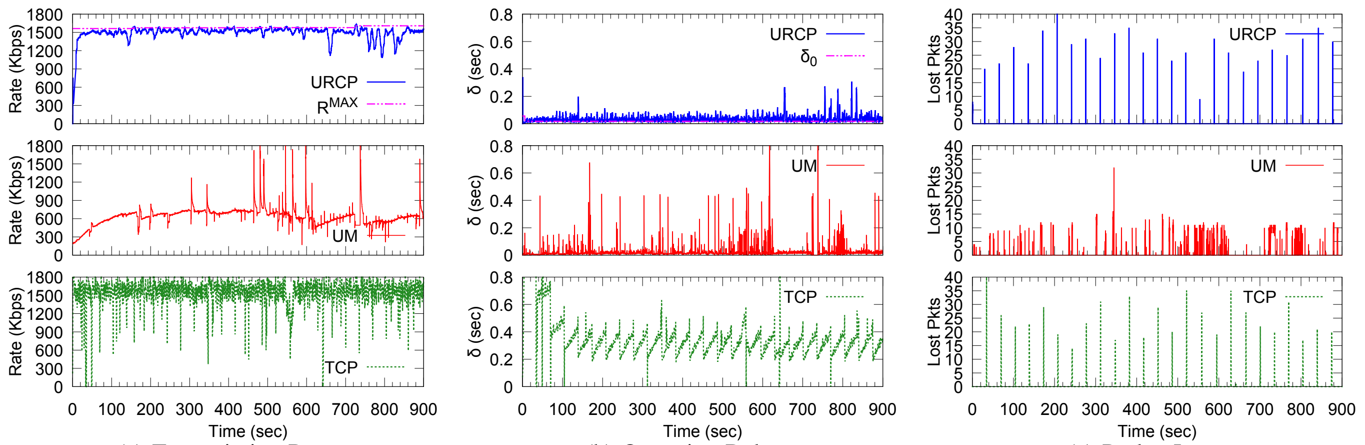


(f) PSNR for Y (luminance)

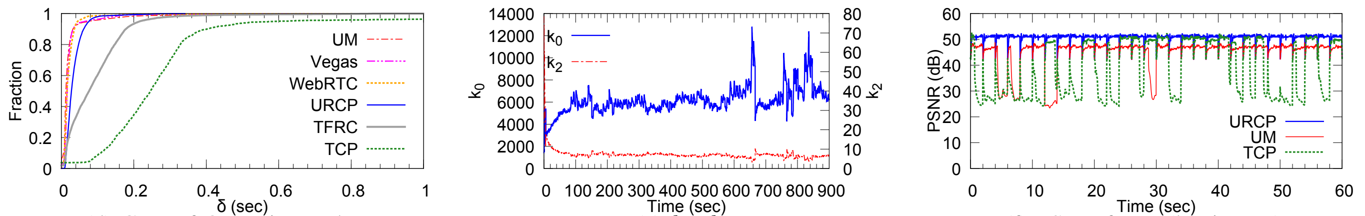
Fig. 3: Results for WiMAX network.

A. *Performance of URCP*

We first show the performance of URCP and compare it with the protocols described above. In Table II, we show the following statistics: receiver rate (throughput), operating delay, 90-th percentile operating delay, packet loss, and the average PSNR of a standard video test sequence ("akiyo" [30]) using the method described. The video sequence has CIF-size resolution (352x288) at 30 fps and is representative of a video conferencing session. In Figs. 3, 4, 5, and 6 we also show the throughput vs. time, operating delay vs. time, lost packets vs. time, CDF of operating delay, and the PSNR vs. time for various protocols

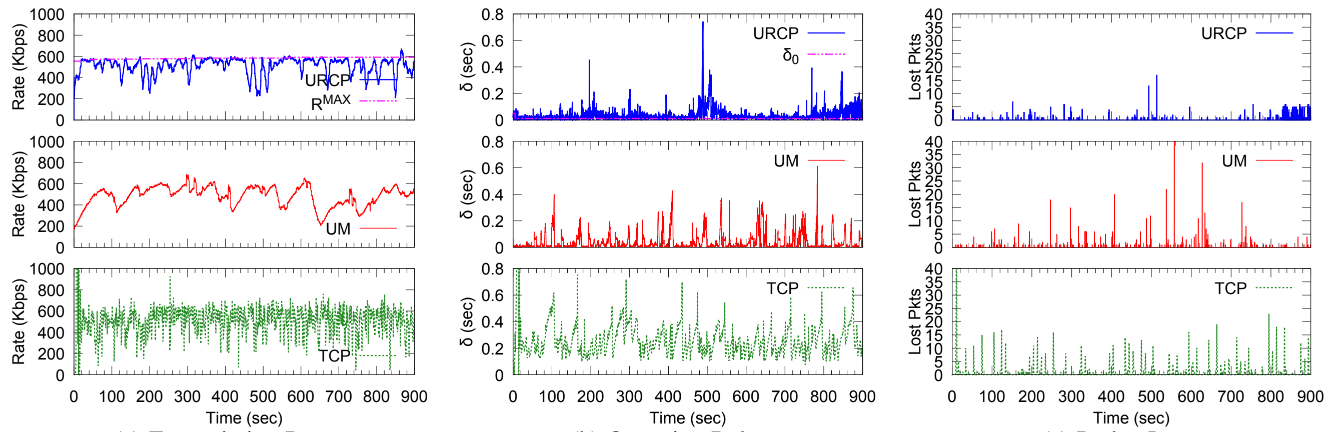


(a) Transmission Rate (b) Operating Delay (c) Packet Loss

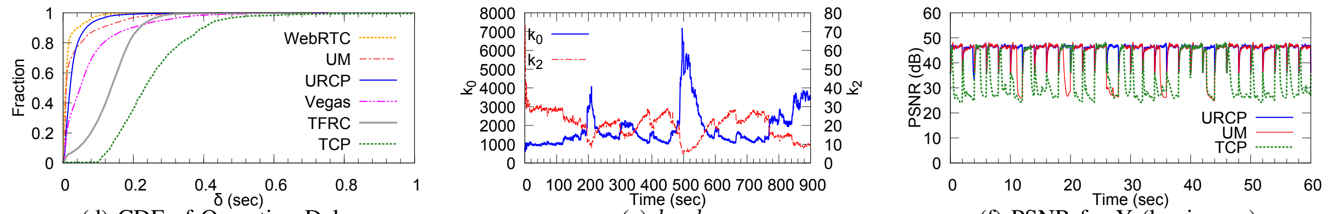


(d) CDF of Operating Delay (e) k_0, k_2 (f) PSNR for Y (luminance)

Fig. 4: Results for 4G (HSPA+) network.



(a) Transmission Rate (b) Operating Delay (c) Packet Loss

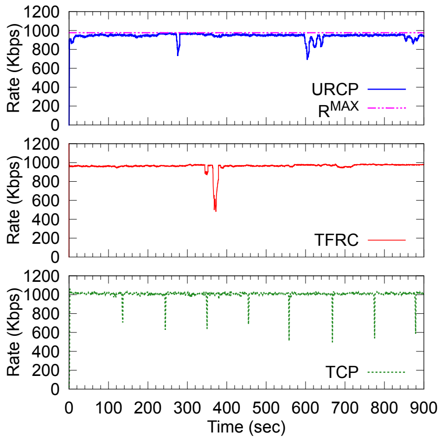


(d) CDF of Operating Delay (e) k_0, k_2 (f) PSNR for Y (luminance)

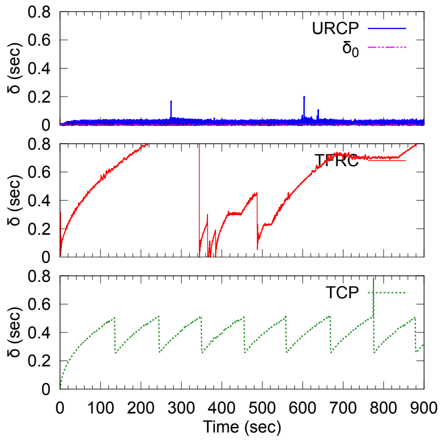
Fig. 5: Results for Wi-Fi network.

for the Clearwire WiMAX, T-Mobile 4G (HSPA+), public Wi-Fi hotspot, and cable modem respectively.

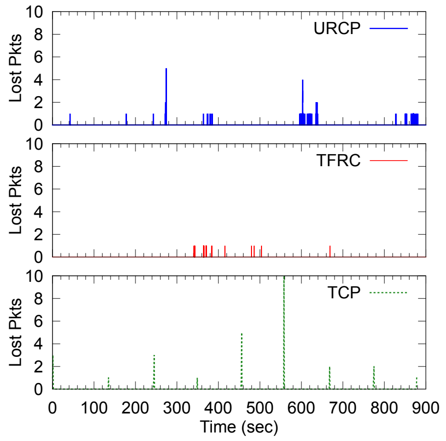
From the results, we see that for all networks URCP is able to achieve similar throughput as the loss based protocols, TCP NewReno and TFRC. However, the operating delay, 90-th percentile operating delay, and packet loss are significantly better. The operating delay is sometimes better by up to two orders of magnitude. This is obvious from the CDF of the operating delay in Figs. 4 and 5. This is expected as loss based protocols target to have full link utilization but at high operating delay levels whereas as URCP attempts to have an operating delay which is close to the inherent delay noise, making the congestion



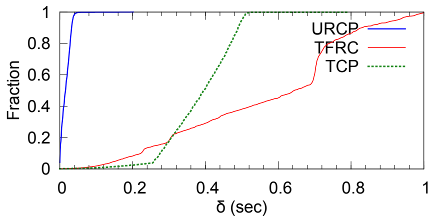
(a) Transmission Rate



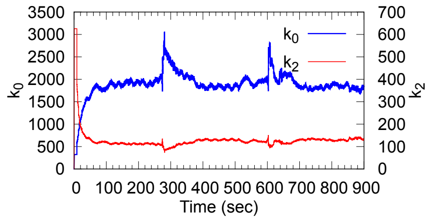
(b) Operating Delay



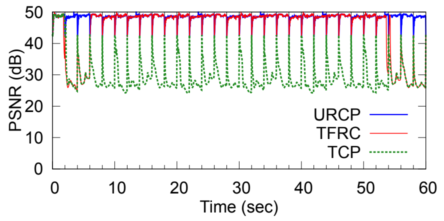
(c) Packet Loss



(d) CDF of Operating Delay



(e) k_0, k_2



(f) PSNR for Y (luminance)

Fig. 6: Results for cable modem network.

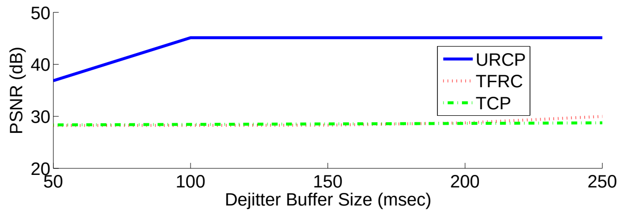
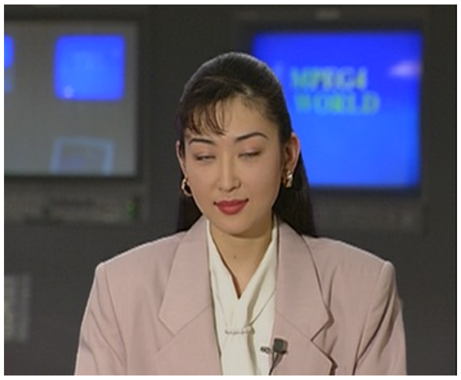
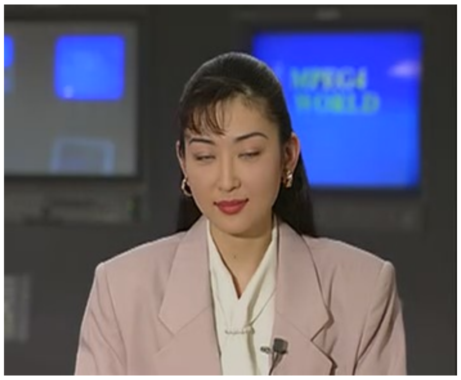


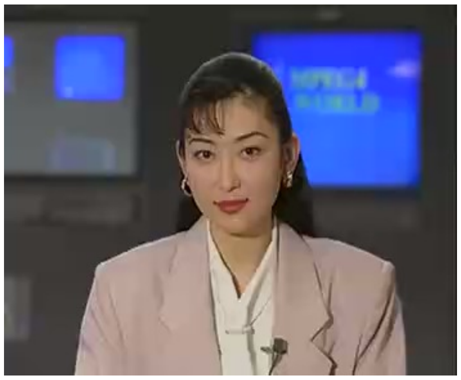
Fig. 7: PSNR performance as a function of dejitter buffer size for the WiMAX network.



(a) Original



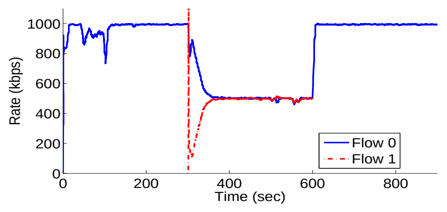
(b) URCP



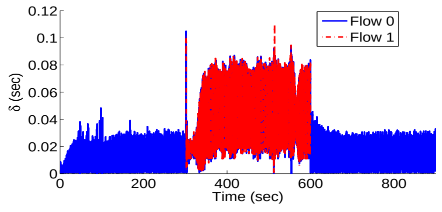
(c) TFRC

Fig. 8: Frames taken from decoded video sequence using various protocols.

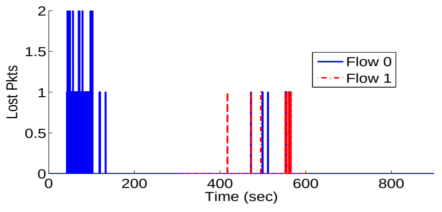
induced queuing delay insignificant. Although TCP NewReno and TFRC have fairly high throughput, when looking at PSNR values from a video conferencing application, both suffer due to the fact that high operating delay results in packets which are considered lost due to late arrival which results in the frames being reconstructed using the error concealment logic present in the decoder.



(a) Transmission Rate

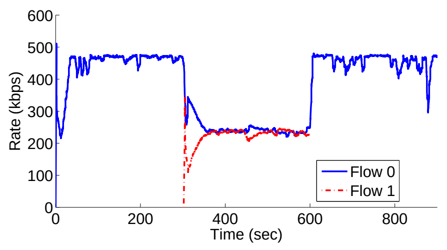


(b) Operating Delay

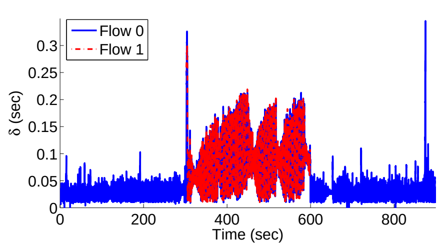


(c) Packet Loss

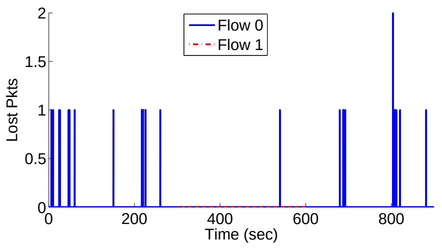
Fig. 9: Results for cable modem network with multiple flows.



(a) Transmission Rate



(b) Operating Delay

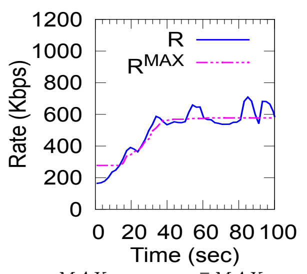


(c) Packet Loss

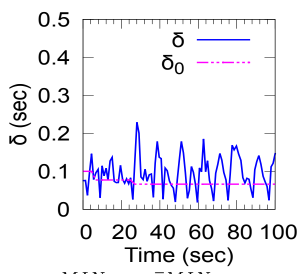
Fig. 10: Results for WiMAX network with multiple flows.

TABLE IV: URCP performance with two URCP flows (first five rows) and URCP plus competing TCP flow (last row).

		Time 0-300sec (Flow 0)				Time 300-600sec (Flow 0 & 1)				Time 600-900sec (Flow 0)			
Network	Flow	Avg. Rate (kbps)	Avg. Delay (msec)	90% Delay (msec)	Loss Rate	Avg. Rate (kbps)	Avg. Delay (msec)	90% Delay (msec)	Loss Rate	Avg. Rate (kbps)	Avg. Delay (msec)	90% Delay (msec)	Loss Rate
ADSL	0	654.90	10	16	0.0019	345.21	20	36	0.0023	692.87	9	12	0.0019
	1	-	-	-	-	398.43	20	35	0.0024	-	-	-	-
Cable Modem	0	969.61	13	24	0.0060	537.73	43	74	0.0011	987.70	15	27	0
	1	-	-	-	-	478.32	45	75	0.0001	-	-	-	-
Ethernet	0	8176.34	0.9	1.2	0.0013	4259.61	2.7	4.1	0.0023	8254.17	1.0	1.5	0.0009
	1	-	-	-	-	4410.91	2.6	4.0	0.0020	-	-	-	-
Wi-Fi	0	502.76	88	176	0.0257	274.31	196	317	0.0315	489.51	77	153	0.0156
	1	-	-	-	-	293.29	195	316	0.0326	-	-	-	-
WiMAX	0	451.98	28	44	0.0008	247.22	94	164	0.0001	455.34	30	47	0.0011
	1	-	-	-	-	225.69	94	164	0	-	-	-	-
Wi-Fi, TCP	URCP	529.13	53	111	0.0132	219.16	232	355	0.0428	489.80	85	170	0.0255
	TCP	-	-	-	-	347.19	267	414	0.0394	-	-	-	-



(a) $R^{MAX} = 0.5\bar{R}^{MAX}$ initially.



(b) $\delta^{MIN} = 2\bar{\delta}^{MIN}$ initially.

Fig. 11: Adaptation of R^{MAX} and δ^{MIN} for Wi-Fi in case they are initially mis-estimated. We see that the parameters are able to self-correct as new values are observed.

When compared to the delay based state-of-art protocols, TCP Vegas, WebRTC, and delay-based UM, we see that URCP is able to maintain similar throughput and operating delay levels on clean networks such as ADSL, Cable Modem, and Ethernet.

However, on noisy networks such as Wi-Fi, WiMAX, and 4G (HSPA+), the throughput with URCP is significantly better than existing delay based protocols, whereas the operating delay and loss levels are similar. This is because URCP adapts the UM framework to have an operating level which allows for full link utilization. The delay seen by all protocols is similar since even if there is no protocol induced congestion, we cannot go below the inherent delay noise. When looking PSNR values for the video conferencing application, the reduced throughput results in poorer PSNR values for TCP Vegas, WebRTC, and delay-based UM.

In conclusion, we see that URCP gives similar throughput as existing loss based protocols, but at much lower operating delay levels. In addition, it gives higher throughput than existing delay based state-of-art techniques on noisy networks.

In Fig. 7, we show the PSNR results as a function of dejitter buffer size for the WiMAX network. We see that we are able to maintain a high PSNR even if the dejitter buffer size reduces to 100ms. Only if it is reduced to 50ms do we see performance degradation.

We also show a representative decoded frame in Fig. 8. If the URCP protocol is used, frames arrive in time and the decoding is similar to the original. If TFRC is used, some frames are lost due to late arriving packets. The error concealment logic in the decoder copies a previous frame which may look very different than the original. In this frame, the newscaster's eyes are different.

In Figs. 4 and 5, we also show parameter adaptation (k_0 and k_2) vs. time for URCP. We also plot the estimated bottleneck link capacity, R^{MAX}, and the estimated operating delay noise, δ^{MIN}. We see that even for noisy networks such as HSPA+ and Wi-Fi, we are able to obtain fairly stable estimates of R^{MAX} and δ^{MIN}.

B. *Effects of Initial Parameter Mis-Estimation*

In Sec. VII-A, the initial parameter estimates for R^{MAX} and δ^{MIN} are good estimates of the capacity and operating delay noise. To study the effects of mis-estimation, we modify the actual initial parameters obtained and analyze the performance. We show results for the four possible mis-estimation cases:

1) Over-estimation of $\bar{R}^{MAX}$: $R^{MAX} = 2\bar{R}^{MAX}$
2) Under-estimation of $\bar{R}^{MAX}$: $R^{MAX} = 0.5\bar{R}^{MAX}$,
3) Over-estimation of $\bar{\delta}^{MIN}$: $\delta^{MIN} = 2\bar{\delta}^{MIN}$,
4) Under-estimation of $\bar{\delta}^{MIN}$: $\delta^{MIN} = 0.5\bar{\delta}^{MIN}$,

The throughput, operating delay, and packet loss results are shown in Table III for several of the networks. In the Table, we also show the initial mis-estimated values of R^{MAX} and δ^{MIN} and show the final values after in-session adaptation. We plot the results for two of the cases, under-estimation of $\bar{R}^{MAX}$ and over-estimation of $\bar{\delta}^{MIN}$, in Fig. 11 for the Wi-Fi network. In the case $\bar{R}^{MAX}$ is under-estimated, we see that R^{MAX} increases as higher rates are seen. Similarly, in the case $\bar{\delta}^{MIN}$ is over-estimated, δ^{MIN} decreases as lower operating delay values are seen.

If $\bar{R}^{MAX}$ is over-estimated, we achieve full throughput at the expense of a slightly higher operating delay. This over-estimation can only be corrected by re-probing the network. However, the effect of the over-estimation is limited since the additional delay due to this grows as $\sqrt{\frac{R^{MAX}}{\bar{R}^{MAX}}}$ as we see from (19) (sub-linear increase).

If $\bar{R}^{MAX}$ is under-estimated, we can still achieve close to full throughput, as R^{MAX} can correct itself (from Eqn. 14) if we see higher rates in the session. From Fig. 11 and Table III, we see that R^{MAX} does increase during the session. In this case, R^{MAX} increases from 420kbps to 822kbps.

If $\bar{\delta}^{MIN}$ is over-estimated, we see that we can achieve full rate and still operate at low congestion levels. This is due to the fact that δ^{MIN} can correct itself if we see lower delay values in the session as we see from Eqn. 15. δ^{MIN} decreases from 10ms to 8ms for the Wi-Fi network. It is not able to see the true value (4-5ms) since we are constantly pushing the network with traffic. In a real case, a video conferencing session may have some quiet periods where it can see lower operating delay noise. The effect of this is that the operating delay is slightly higher than if it was estimated correctly.

If $\bar{\delta}^{MIN}$ is under-estimated, we may under-utilize the link slightly as the only way to correct for under-estimation of δ^{MIN} is to re-probe the network. However, as we see from (15), since we adjust the operating congestion level using *observed* operating delay, δ^{AVE}, the affect of under-estimation is mitigated (the rate only drops from 750kbps to 600kbps).

C. Multiple URCP Flows

We examine the performance of URCP when multiple flows are sharing a bottleneck link. We consider the case when one flow is sending from 0-900 seconds and the other flow from 300-600 seconds. The overall performance of this is summarized in Table IV for several of the networks (first five rows of the table). The performance is also plotted in Fig 10 and 9 for the WiMAX and cable modem network respectively. We note that URCP is able to fairly share the bottleneck link when multiple URCP flows are present. Although the operating delay does increase when two flows are present (as predicted from (19)), it reduces once the second flow departs.

D. Competing TCP Flows

We also study the case when a competing TCP flow joins the network and present the result in the last row of Table IV. We run a URCP flow from 0-900 seconds and a TCP flow joins from 300-600 seconds. We run this experiment on the Wi-Fi network. When the TCP flow joins, the average rate reduces from 529kbps to 219kbps with the TCP flow taking the remaining rate. The average delay increases from 53msec to 232ms and the loss rate increases from .01 to .04. Once the TCP flow leaves, both delay and loss return to original levels. This shows that we are able to effectively compete with TCP flows while still being able to return to low operating congestion levels.

VIII. Final Remarks

In this paper, we have shown that existing rate control protocols either deliver high throughput with high operating delay and loss or low throughput with low operating delay and loss on *noisy* networks. Since RTC applications require high throughput while maintaining low operating delay and loss levels, we developed a novel Universal Rate Control Protocol (URCP) which is able to achieve the desired characteristics on a range of networks include noisy ones such as Wi-Fi, WiMAX, and 3G/4G.

The key insight in URCP is that parameter adaptation based on stable and relatively easy to estimate network characteristics can be used to modify the *operating delay and loss point* of the rate control protocol so that *congestion induced delay and loss* can be minimized while still being able to achieve the full bottleneck link capacity.

References

[1] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. DCTCP: Efficient packet transport for the commoditized data center. In *Proc. of SIGCOMM*. ACM, 2010.

[2] M. Borri, C. Finelli, M. Gerla, and M. Merani. Adaptive GAIMD with binary decision: A novel congestion control approach for the internet. In *IEEE International Conference on Communications (ICC)*, volume 1, pages 152–157, june 2006.

[3] L. S. Brakmo and L. L. Peterson. TCP Vegas: end-to-end congestion avoidance on a global internet. *IEEE Journal on Selected Areas In Communication*, 13(8):1465–1480, Oct. 1995.

[4] M. Chen. *A General Framework for Flow Control in Wireless Networks*. PhD thesis, University of California at Berkeley, Berkeley, CA 94706, Dec. 2006.

[5] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. Chou. Utility maximization in peer-to-peer systems. In *ACM SIGMETRICS*, jun 2008.

[6] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle. Layering as optimization decomposition:a mathematical theory of network architectures. *Proceedings of the IEEE*, 95(1):255–312, Jan. 2007.

[7] S. Floyd and T. Henderson. *The NewReno Modification to TCP's Fast Recovery Algorithm*, Apr. 1999. RFC 2582.

[8] H.264/AVC JM Reference Software Download. http://iphome.hhi.de/suehring/tml/download.

[9] S. Ha, I. Rhee, and L. Xu. CUBIC: A new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating System Review*, 42(5):64–74, Jul 2008.

[10] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP friendly rate control (TFRC): Protocol specification. *RFC-3448*, Jan 2003.

[11] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. A close examinization of performance and power characteristics of 4G LTE networks. In *ACM Mobisys*, June 2012.

[12] V. Jacobson. Congestion avoidance and control. In *Proc. ACM SIGCOMM*, pages 314–329, Stanford, CA, Aug. 1988.

[13] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput. *IEEE/ACM Trans. Networking*, 11:537–549, Aug. 2003.

[14] M. Jain and C. Dovrolis. Ten fallacies and pitfalls on end-to-end available bandwidth estimation. In *IMC*, 2004.

[15] F. P. Kelly. Fairness and stability of end-to-end congestion control. *European Journal of Control*, 9:159–176, 2003.

[16] F. P. Kelly, A. Maulloo, and D. Tan. Rate control for communication networks: shadow prices, proportional fairness, and stability. *Journal of the Operationl Research Society*, 49:237–252, 1998.

[17] S. Lee, I. Pefkianakis, A. Meyerson, S. Xu, and S. Lu. Proportional fair frequency-domain packet scheduling for 3gpp lte uplink. In *INFOCOM 2009, IEEE*, pages 2611–2615. IEEE, 2009.

[18] S. H. Low and D. E. Lapsley. Optimization flow control, I: Basic algorithm and convergence. *IEEE/ACM Trans. Networking*, 7(6):861–875, Dec. 1999.

[19] H. Lundin, S. Holmer, and H. Alvestrand. IETF Internet Draft: A Google congestion control algorithm for real-time communication on the world wide web. http://tools.ietf.org/id/draft-alvestrand-rtcweb-congestion-02.txt.

[20] A. Maor and Y. Mansour. AdaVegas: Adaptive control for TCP Vegas. In *Proc. of IEEE Globecom*, volume 7, pages 3647–3651, Dec. 2003.

[21] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. In *SIGCOMM Comput. Commun. Rev.*, volume 27, pages 67–82, New York, NY, USA, July 1997. ACM.

[22] S. Mehrotra, H. Chen, S. Jain, J. Li, B. Li, and M. Chen. Bandwidth management for mobile media delivery. In *Proc. of IEEE Globecom*. IEEE, Dec. 2012.

[23] S. Moon, P. Skelly, and D. Towsley. Estimation and removal of clock skew from network delay measurements. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 227–234. IEEE, 1999.

[24] Linux kernel 3.7 tcp vegas implementation. http://lxr.free-electrons.com/source/net/ipv4/tcp_vegas.c.

[25] Rtcweb status pages. http://tools.ietf.org/wg/rtcweb.

[26] C. B. Samios and M. K. Vernon. Modeling the throughput of TCP Vegas. In *ACM SIGMETRICS*, pages 71–81, New York, NY, USA, 2003. ACM.

[27] G. Song and Y. Li. Utility-based resource allocation and scheduling in OFDM-based wireless broadband networks. *IEEE Computer Magazine*, pages 127 – 134, Dec. 2005.

[28] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *IMC*, Oct 2003.

[29] K. Winstein, A. Sivaraman, and H. Balakrishnan. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *NSDI*. USENIX, 2013.

[30] Xiph.org Test Media. http://media.xiph.org/video/derf.

[31] Y. Yang and S. Lam. General aimd congestion control. In *International Conference on Network Protocols*, pages 187–198, 2000.