PROTEUS: Network Performance Forecast for Real-Time, Interactive Mobile Applications

Qiang Xu
University of Michigan
Ann Arbor, MI
qiangxu@umich.edu

Sanjeev Mehrotra
Microsoft Research
Redmond, WA
sanjeevm@microsoft.com

Z. Morley Mao
University of Michigan
Ann Arbor, MI
zmao@umich.edu

Jin Li
Microsoft Research
Redmond, WA
jinl@microsoft.com

ABSTRACT

Real-time communication (RTC) applications such as VoIP, video conferencing, and online gaming are flourishing. To adapt and deliver good performance, these applications require accurate estimations of short-term network performance metrics, e.g., loss rate, one-way delay, and throughput. However, the wide variation in mobile cellular network performance makes running RTC applications on these networks problematic. To address this issue, various performance adaptation techniques have been proposed, but one common problem of such techniques is that they only adjust application behavior reactively after performance degradation is visible. Thus, proactive adaptation based on accurate short-term, fine-grained network performance prediction can be a preferred alternative that benefits RTC applications.

In this study, we show that forecasting the short-term performance in cellular networks is possible in part due to the channel estimation scheme on the device and the radio resource scheduling algorithm at the base station. We develop a system interface called PROTEUS, which passively collects current network performance, such as throughput, loss, and one-way delay, and then uses regression trees to forecast future network performance. PROTEUS successfully predicts the occurrence of packet loss within a 0.5s time window for 98% of the time windows and the occurrence of long one-way delay for 97% of the time windows. We also demonstrate how PROTEUS can be integrated with RTC applications to significantly improve the perceptual quality. In particular, we increase the peak signal-to-noise ratio of a video conferencing application by up to 15dB and reduce the perceptual delay in a gaming application by up to 4s.

MobiSys'13, June 25–28, 2013, Taipei, Taiwan

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication; C.2.4 [**Distributed Systems**]: Client/server; C.4 [**PERFORMANCE OF SYSTEMS**]: Measurement techniques; C.4 [**PERFORMANCE OF SYSTEMS**]: Modeling techniques

General Terms

Design, Measurement, Performance

Keywords

Cellular network prediction; Interactive mobile applications; Real-time communication (RTC) applications

1. INTRODUCTION

One of the biggest trends today is the accelerated adoption of mobile devices, whose power is approaching that of PCs. The rapid adoption of mobile devices is resulting in the migration of real-time communication (RTC) applications from PCs to mobile devices. RTC applications include (i) multimedia real-time communication including VoIP/video conferencing applications such as Skype, FaceTime, and Google+ Hangout; (ii) interactive multi-player gaming applications such as Draw Something, Modern Combat 3, and Call of Duty; and (iii) application sharing, desktop sharing, and virtual desktop interface (VDI). RTCWeb (Real-Time Collaboration over Web) is an ongoing effort to enable RTC applications to run inside browsers without plug-ins [8].

Running RTC applications in mobile networks presents a major challenge since they require accurate short-term estimates of network metrics, e.g., loss rate, one-way delay (OWD), and throughput, whereas cellular network performance varies rapidly and widely [20, 24, 44]. Although performance adaptation solutions have been proposed [1, 2, 5, 7, 34, 40, 48], they are ineffective for RTC applications, where the end-to-end delay between content generation and content consumption is on the order of network latencies. Thus, techniques such as packet retransmissions or use of large client-side de-jitter buffers to absorb jitter variations, effective for web browsing and video playback, will lead to intolerable delays in RTC applications.

Instead, RTC applications need to rely on an appropriate amount of forward error correction (FEC) to account for packet loss and a small yet appropriately sized de-jitter buffer to absorb varying packet delay. Over-protection using FEC wastes precious bandwidth, while under-protection negates the effectiveness of FEC, as any unrecoverable packet loss will lead to severe quality degradation in voice calling and video conferencing. Similarly, if the de-jitter buffer size is too large, it adds to the application latency. If it is too small, late arriving packets will be considered lost, which also leads to significant quality degradation. Accurate throughput prediction is also necessary to allow bandwidth intensive applications to maximize throughput without incurring additional queuing delay. Thus, short-term, fine-grained network performance prediction on mobile networks is of great importance.

Existing work on mobile network performance measurement and prediction is of limited importance to RTC applications for several reasons: (i) the measurement or prediction approaches are carried out on coarse-grained time granularity or offline, which are not suitable for RTC applications; (ii) the performance adaptation techniques react to performance only after observing any degradation, when the user experience has already been negatively impacted; and (iii) the existing techniques used by RTC applications for network prediction are usually naïve such as using the previous performance,the average of previous performance, or linear regression techniques, which work well only for relatively stable wired networks.

In this study, we propose a system interface named PROTEUS designed to accurately predict fine-grained detailed network performance in real time over short time scales allowing applications to adjust their behavior to best optimize their performance. The fine-grained and accurate performance prediction in real time is necessary yet challenging from three aspects: (i) cellular network performance may be affected by a multitude of unobservable factors, such as radio resource scheduling, other users sharing the spectrum, and signal-to-noise ratio; (ii) network performance predictions on fine-grained time granularity is inherently noisier than coarse-grained predictions; and (iii) active probing is prohibitive for RTC applications due to significant resource consumption, while passive monitoring has limited visibility only based on traffic generated by applications.

To address the above challenges, PROTEUS utilizes a machine learning framework based on regression trees to learn the trend of network performance over short, fine-grained time windows using previous available observations. Even though past work has provided the evidence that cellular networks have a potential to have predictability for network performance, to our best knowledge, PROTEUS is the first system to forecast short-term network performance in real time, and the first to demonstrate the benefit of proactively adjusting the operating parameters in real time for use by mobile RTC applications. The key contributions of our paper are as follows:

- We characterize short-term network behavior and investigate the performance predictability in three major cellular networks of AT&T, T-Mobile, and Sprint based on more than 400 hours of traces across 3 locations. We identify the existence of strong predictability over short time scales for these cellular networks.
- We discover that loss rate, one-way delay, and throughput in cellular networks can be accurately predicted in real time from past measurements by feeding past network performance metrics into regression trees. For a following 0.5s time window, we can predict loss occurrence with 98% accuracy, late packet arrival with 97% accuracy, and throughput with a median error of 10kbps with average throughput ranging from 100–800kbps.
- We prototype PROTEUS, an efficient framework that forecasts achievable network performance in real time and evaluate its usage by a video conferencing system and an online game, i.e., Draw Something. By adapting its behavior through prediction from PROTEUS, the video conferencing system can improve its peak signal-to-noise ratio by up to 15dB over the state-of-the-art adaptation techniques. Utilizing PROTEUS, Draw Something can reduce the perceptual delay in viewing the stroke-by-stroke animation by up to 4s.

The rest of this paper is organized as follows. §2 identifies the potential factors that allow for short-term, fine-grained performance predictability in cellular networks. §3 overviews the infrastructure of PROTEUS, followed by §4 where we characterize short-term predictability of cellular networks. §5 applies PROTEUS into a video conferencing system and a multi-player gaming system and evaluates the benefits from PROTEUS. The related studies are discussed in §6 and we conclude our paper in §7.

2. CELLULAR NETWORK BACKGROUND

In this section, we examine the relevant network behavior in cellular networks to establish the time scale in which network performance may be predictable.

In UMTS, EV-DO, and LTE networks, a mobile device estimates the channel quality through the perceived signal-to-noise ratio of the pilot signal from the base station in every time slot, e.g., 1.67ms for EV-DO. The device determines the modulation and coding scheme (i.e., DRC) from the channel quality and reports back to the base station. Depending on the channel quality, the base station allocates time slots via proportional fair scheduling to fairly allocate radio resource while maximizing the overall radio resource utilization [18].

To precisely understand how the proportional fair scheduler works, let $\mathtt{W}[\mathtt{n}]$ denote the exponentially averaged throughput at time slot $\mathtt{n}$ which is computed from $\mathtt{W}[\mathtt{n}] = (1-\alpha)\mathtt{W}[\mathtt{n}-1] + \alpha \cdot \mathtt{I}[\mathtt{n}-1] \cdot \mathtt{C}[\mathtt{n}-1]$, where $\mathtt{I}[\cdot]$ is the indicator function on whether the user is served in a given time slot, $\mathtt{C}[\cdot]$ is the function of channel quality report, and α is the discount factor controlling the aggressiveness for the base station switching time slots across users [28]. The proportional fair scheduler allocates time slot $\mathtt{n}$ to a user with the maximum value of $\frac{\mathtt{C}[\mathtt{n}]}{\mathtt{W}[\mathtt{n}]}$ to balance serving between the device with the best channel quality and the device consuming the most resources.

The average duration of staying in a DRC is hundreds to thousands of time slots for stationary devices and tens of time slots for moving devices [31]. Moreover, a device spends a large fraction of time in one dominant DRC, indicating the presence of a dominant channel condition [31]. Thus, the number of continuous time slots that a device can occupy is decided by $\frac{1}{\mathtt{W}[\mathtt{n}]}$, which is affected by the discount factor α. To encourage a device to access a time slot in time [21] and to occupy enough continuous time slots to efficiently deliver a non-trivial amount of data, α is usually

set to a small value (e.g., 0.001) [25]. This allows a device to occupy the order of $\approx \frac{1}{\alpha} = 1000$ time slots, which means that a device can occupy the channel with the same DRC on the order of ≈1.67s. Since the network performance on a wireless link between the device and base station is largely a function of the channel condition and the DRC value, we expect that it should be possible to predict network performance on a similar time scale, i.e., 1.67s. Note that we expect the presence of network predictability in other variants of 3G or 4G networks as well due to the consistent utilization of proportional fair scheduling and channel quality report schemes [27].

3. PROTEUS OVERVIEW

PROTEUS is designed to complement the underlying transport protocol and any performance optimization techniques (e.g., FEC, de-jitter buffers, congestion control) that are widely used by RTC applications. For example, video conferencing applications using UDP have their own congestion control based on delay and loss. Applications using TCP rely on TCP's congestion control algorithm. The goal of PROTEUS is to provide applications with prediction of delay, loss, and throughput, so that they can perform application-specific optimizations such as tuning source bit-rate, amount of FEC inserted, and de-jitter buffer size.

To avoid measurement overhead, PROTEUS does not actively probe the network. Instead, PROTEUS is designed as a library that relies on application traffic to collect network measurements. This is done by appending and recording the time and sequence numbers for outgoing and incoming packets. PROTEUS can be either in user space or inside the kernel. Inside the kernel, PROTEUS has lower overhead. As shown in Figure 1, applications (e.g., `app #1`) can call `send'()` and `recv'()`, which are PROTEUS's wrapped version of standard socket API's `send()` and `recv()`. The socket wrapper allows PROTEUS to transparently collect network statistics (one-way delay, loss, and throughput) by inserting sequencing and timing information into packet headers. However, there may be many applications that already append sequencing and timing information in their packet headers to compute similar network statistics, e.g., RTP packets used by many VoIP and video conferencing applications already contain sufficient information to accurately compute network statistics. These applications (e.g., app #2) can alternatively use the standard socket APIs and inform PROTEUS with network statistics.

PROTEUS provides applications and the socket wrapper another two sets of APIs: the inform API and the inquiry API. The inform API enables the socket wrapper (e.g., `app #1`) and applications (e.g., `app #2`) to inform PROTEUS with packet sequencing and timing information, whereas the application issues the inquiry API to obtain from PROTEUS the predicted future network performance, as depicted in Figure 1. Based on this fine-grained information of packet sequencing and timing provided by either the wrapped socket (e.g., `app #1`) or the application (e.g., `app #2`) through the inform API, PROTEUS can compute current packet loss rate, one-way delay, and throughput from which it can predict future network performance. PROTEUS uses a clock drift compensation technique [35] to adjust the remote timestamp to the local clock to compute one-way delay.

The overhead of adding timestamps and sequence numbers into packet headers is acceptable, which has been evaluated in previous research [3, 41]. The statistics of loss, delay, and throughput are collected over small non-overlapping time slices (i.e., time windows). This is the granularity over which statistics collection as well prediction are performed. Using the observed statistics from a certain number of previous time windows (i.e., information windows), PROTEUS trains regression trees to identify the dependencies across various network metrics and to forecast the network performance in future time windows. There could be more than one way to learn traffic histories. We resort to regression trees simply as it is one of the state-of-the-art approaches that have relatively low memory and computation overhead [17]. Another advantage of regression trees over other techniques is that it can provide a real-value prediction of a metric as opposed to simply a binary decision.

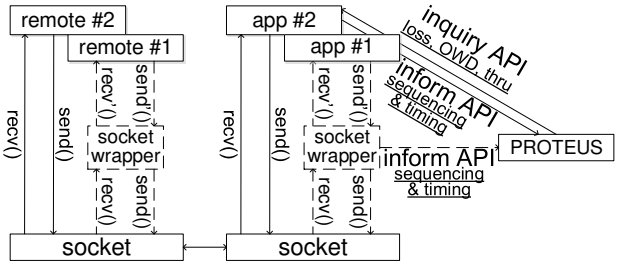


Figure 1: The overall design of PROTEUS. The details of its interaction with an application are shown in Figure 11.

The accuracy of PROTEUS's prediction is determined by two factors: (i) the inherent predictability of cellular network performance (described in §4.1), and (ii) the effectiveness of PROTEUS's forecasting technique (examined in §4.2).

The information exchanged through the inform API between applications and PROTEUS is common across applications. However, the information from the inquiry API and the resulting action taken by each application may be very different. In Table 1, we classify applications into four broad categories depending on how sensitive their performance is to the three network parameters of packet loss, one-way delay, and throughput and show how PROTEUS can be useful.

- Type 1 consists of applications whose performance is not sensitive to loss, one-way delay, or throughput. These are either non-real-time, low-bandwidth applications insensitive to network performance, or applications that do not benefit much from prediction. For example, the performance of Dropbox and other file transfer applications depends on throughput, but there is little adaptation in response to throughput prediction.
- Type 2 consists of applications whose performance is not sensitive to packet loss or one-way delay but is sensitive to throughput. These are non-real-time, high-bandwidth applications such as video on demand (VOD). Such applications can take steps to improve performance such as adapting the rate of the stream served to the client [5] in response to throughput prediction.
- Type 3 consists of applications whose performance is sensitive to packet loss and/or one-way delay, but not to throughput. These are real-time, interactive, low-bandwidth applications such as VoIP. For these applications, throughput prediction is of little benefit. However, loss prediction can be

type	description	example	inquiry API
1	non-real-time, low rate, or insensitive to performance	Weather, Calender, Reminder, Dropbox	N/A
2	sensitive to throughput	YouTube, NetFlix. Hulu	throughput
3	sensitive to loss and one-way delay (OWD)	Skype Call, Google Voice	loss, OWD
4	sensitive to loss, OWD, and throughput	Skype Video, Google+ Hangouts, Draw Something, online games	loss, OWD, throughput

Table 1: The types of applications based on the sensitivity to network loss, OWD, and throughput.

used to control the amount of FEC used [15]. The one-way delay prediction can be used to adjust the de-jitter buffer size for adaptive playout to reduce both late arrival induced loss [15] and de-jitter buffer induced delay.

• Type 4 consists of applications whose performance is sensitive to packet loss, one-way delay, and throughput prediction. These are real-time, interactive, high-bandwidth applications such as video conferencing, and real-time software applications such as desktop sharing. PROTEUS can be used for throughput prediction to control the source rate produced by the source. The number of FEC packets to insert can be inferred from loss prediction, and the de-jitter buffer size can be determined from one-way delay prediction.

Type 3 and Type 4 are real-time, interactive applications that can benefit from PROTEUS. Type 2 consists of *non-real-time* applications that can still benefit from throughput prediction.Although a coarse-grained predictor may work for Type 2 applications, it may not work for RTC applications. A coarse-grained predictor can be built upon a fine-grained predictor such as PROTEUS by simply averaging the prediction over multiple time windows.

Types 2, 3, and 4 applications are becoming increasingly popular and contribute to a majority of the traffic on the Internet [47]. This trend is readily visible by the increasing popularity of Skype, FaceTime, and Google+ Hangouts. The increasing push towards the cloud also makes other RTC applications such as online gaming and VDI (virtual desktop infrastructure) scenarios even more important. The broad industry interest in the performance of such applications is apparent from recent efforts around RTCWeb [8] and DASH (Dynamic Adaptive Streaming over HTTP) [6].

4. PREDICTION USING PROTEUS

In this section, we investigate the predictability of cellular network performance and evaluate the accuracy of PROTEUS's forecasting.

4.1 Characterizing Cellular Networks

Although the understanding of the channel estimation scheme on mobile devices and the radio resource scheduling algorithm at base stations, along with the performance predictability observations from previous studies [31] give us confidence that cellular network performance is predictable, it is still uncertain (i) how much predictability there is in cellular networks, particularly in the short term as is needed by RTC applications; (ii) what network features can be the best predictors; and (iii) how much overhead is required to achieve accurate prediction. These challenges have not been fully addressed by any previous research, but need to be studied to improve the performance of Type 3 and Type 4 applications in Table 1.

Autocorrelation for the same network metric. To quantify the predictability of cellular network performance, we first study the correlation across network metrics over time. A high correlation of a particular network characteristic would indicate that the feature should be predictable whereas a low correlation would imply difficulty in prediction. We take network measurements across non-overlapping time intervals, i.e., aforementioned time windows, and then compute the correlation coefficient between current time window with a time window $\mathtt{t}$ in the past, i.e., the time lag, using $\mathtt{R_t} = \frac{\mathtt{E[P_0 \cdot P_t]} - \mathtt{E[P_0]} \cdot \mathtt{E[P_t]}}{\sigma[\mathtt{P_0}] \cdot \sigma[\mathtt{P_t}]}$, where $\mathtt{E}[.]$ is the expectation operation, $\sigma[.]$ is the standard deviation operation, and $\mathtt{P}$ is the stochastic process of any network performance metric[1]. A high autocorrelation coefficient would indicate that the metric is predictable using a linear predictor.

device	carrier	network	down.[1]	up.[1]
iPhone 4	AT&T	UMTS	77(4)	42(4)
Captivate	AT&T	UMTS	159(N/A)	72(N/A)
Atrix	AT&T	HSPA	42(N/A)	20(N/A)
Nexus	T-Mobile	HSPA+	109(N/A)	108(N/A)
dongle	Sprint	EV-DO	44(N/A)	115(N/A)

[1] The number of UDP(TCP) flows, each lasting ≥1 hour.

Table 2: The devices and cellular networks covered by the experiments. FlowSet Refers to the AT&T flows.

To study if cellular network performance can be predicted using fine-grained time windows, e.g., 0.5s, we collect a large number of traces from various cellular networks in Table 2 and compute the autocorrelation of throughput using each trace choosing a time window of 0.5s, i.e., $\mathtt{tw} = 0.5$. We plot the distribution of the autocorrelation coefficient as a function of the time lag in Figure 2(a) for the flows in FlowSet, as defined in Table 2. Given a time lag, we show the distribution across flows via the 5th, 25th, 50th, 75th, and 95th percentiles.

In Figure 2(a), more than half of the time, the autocorrelation coefficient at a time lag of 0.5s is more than 0.6, indicating that the network performance using a time window of size 0.5s is indeed predictable to some degrees using previous time windows. From Figure 2(a), we also see that once the time lag is above 20s, the median autocorrelation coefficient is close to 0, and the 25-75th percentiles are in the range of [-0.05, 0.05]. Therefore, we expect that using the time windows that are within the last 20s is sufficient to predict network performance. The results presented here show the autocorrelation function for throughput. We see similar behavior for packet loss and one-way delay.

Time granularity of performance prediction. One parameter affecting the autocorrelation coefficient is the size of the time window. If the time window chosen is small,

[1]The value of the autocorrelation function lies in the range $[-1, 1]$, with 1 indicating perfect correlation, -1 indicating perfect anti-correlation, and 0 indicating no correlation.

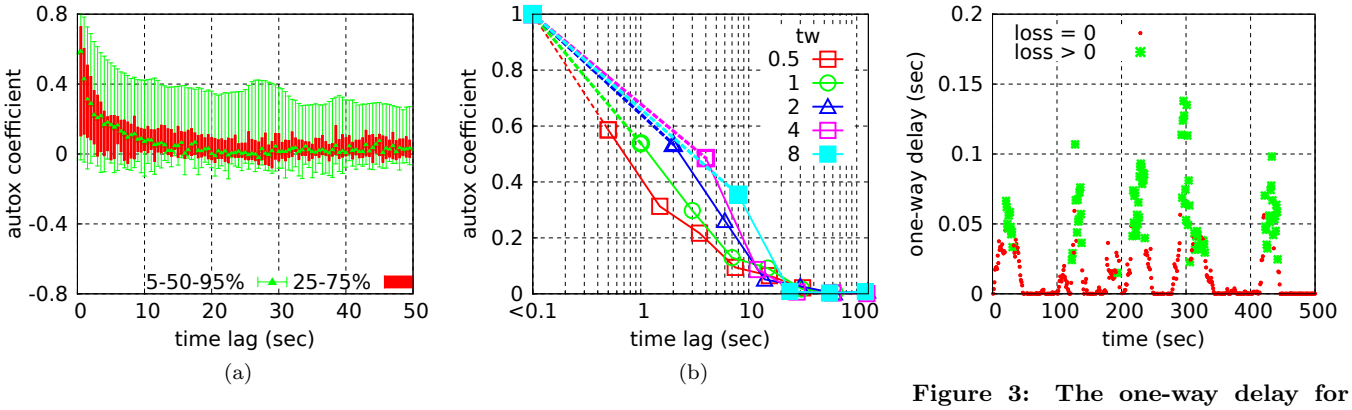


Figure 2: The autocorrelation coefficient of throughput under the impact of (a) the time lag and (b) the time window size ("tw").

Figure 3: The one-way delay for the time windows with and without loss.

the measurements may show high variability, which may be more difficult to predict. However, if the time window chosen is large, the measurements may be affected by long term drift and cannot accurately reflect the fine-grained performance needed by RTC applications. To appropriately select a time window size, we study the effect of time window size on the autocorrelation function. Figure 2(b) shows the median autocorrelation coefficient for UDP throughput as a function of the time lag for various time window sizes, i.e., 0.5s, 1s, 2s, 4s, and 8s. We expect that as the time window size increases, the autocorrelation coefficient for a given time lag should also increase until it eventually converges. In Figure 2(b), "`tw` = 4" being close to "`tw` = 8" indicates that using time windows greater than 4s provides no additional benefit. However, we see that even with a window size of "tw = 0.5", the average autocorrelation coefficient at a time lag of 0.5s is above 0.6 and is similar to that using a larger time window. Thus, we conclude that we can indeed predict network performance from previous time windows using fine-grained windows, e.g., 0.5s, as needed by RTC applications.

In total, we have three observations from studying the autocorrelation function: (i) the current network performance is correlated with the network performance in the previous time windows; (ii) to predict the network performance, the time window size can be as short as 0.5s as required by RTC applications; and (iii) using information within the previous 20s should be sufficient for accurate prediction.

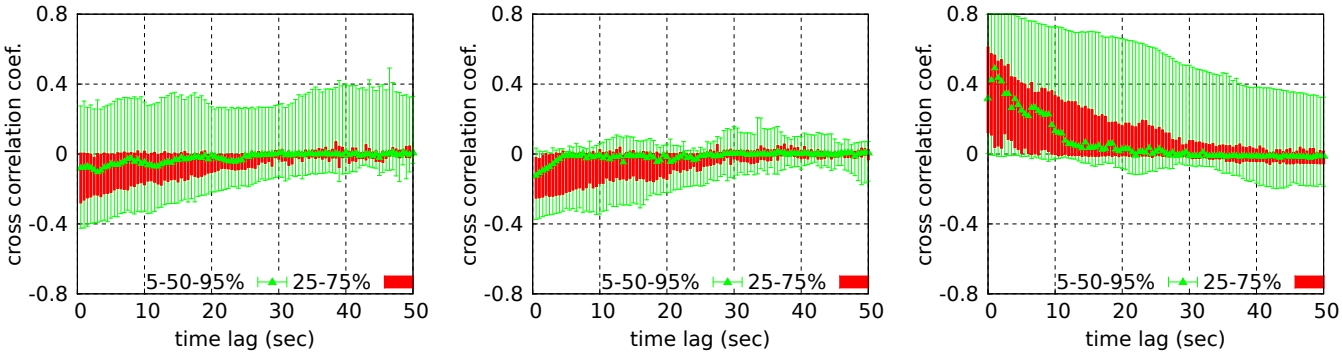
Cross correlation between network metrics. These three observations answer the question of whether network performance is predictable on cellular networks using previous values of a particular metric to predict future values of the same metric. However, the question of whether previous values of other metrics can also be used to improve prediction remains. To study this, we study the cross-correlation coefficient across various different network metrics.

Since packet loss and end-to-end network delay are commonly used as congestion signals by congestion control protocols as well indicate overbuffering in cellular networks [26], we expect that these metrics may be correlated. To confirm this understanding, we investigate the cross correlation between the various network performance metrics. The distribution of the cross-correlation coefficient as a function of the time lag is shown in Figure 4. From Figures 4(a) and (b), we find that throughput does not correlate well with either loss or one-way delay. The maximum absolute median cross correlation coefficient between throughput and loss is less than 0.1, and between throughput and one-way delay is less than 0.15. We believe that this is because the throughput is decided more by the DRC on the link between the device and the base station, while loss and one-way delay are affected more by the congestion along the path. However, from Figure 4(c), we see that loss rate and one-way delay do have a strong correlation with each other. When the time lag is close to 0, the median cross correlation coefficient is around 0.5. Figure 3 further shows this correlation by showing the average one-way delay for each time window as a function of time for time windows with loss and those without loss. We clearly see that for those windows where loss occurs, the average one-way delay is also significantly higher than for those where there is no loss. As a result, the forecasts of loss and one-way delay use both previous loss rate and one-way delay, while throughput prediction uses just the throughput from previous time windows.

4.2 Constructing the Regression Trees

We utilize a framework based on regression trees to automatically learn the correlation between previous network performance and current network performance [17]. Note that there could be more than one way to learn traffic histories and predict network performance, e.g., Markov chains. We resort to regression trees simply because it is one of the state-of-the-art approaches that has relatively low memory and computation overhead. We have tried to use a Markov chain to model the short-term network performance behavior, however, tuning the parameters for the model is ad hoc.

We construct a regression tree for each of the network metrics, with the target of the tree being the metric predicted and the attributes being the metrics used to perform the prediction. The loss regression tree predicts the loss rate in the next time window using the loss rate and one-way delay values from previous time windows. The delay regression tree uses the same attributes to predict the one-way delay in the next time window. The throughput regression tree predicts the throughput in the next time window using the throughput values from previous time windows and the sending rate from the immediate previous time window.



(a) throughput and loss rate. (b) throughput and one-way delay. (c) loss rate and one-way delay.

Figure 4: The cross correlation coefficient between performance metrics. The cross correlation is computed per flow and the distribution across flows of the correlation coefficient under a given time lag is presented as 5-, 25-, 50-, 75-, and 95- percentiles. The correlation coefficient is normalized so that the maximum value is 1 for each flow.

The sending rate is included to address the case in which the application is sending at a rate much below the throughput.

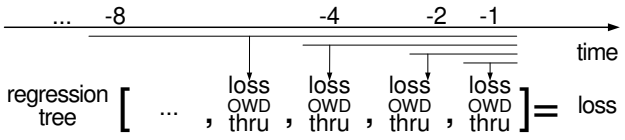


Figure 5: The attributes of the loss regression.

The number of attributes in the regression trees can be controlled by selecting an appropriate information window size from the correlation analysis (Figures 2 and 4). As illustrated by Figure 5, the attributes are computed using time windows in an exponential backoff fashion, i.e., the attributes are the average values of a given metric in the previous $1, 2, 4, 8, \ldots, 2^{\lceil \log_2(\mathtt{M}) \rceil}$ time windows, where $\mathtt{M}$ is the information window size. Creating attributes using this approach allows us to place greater importance on the more recent time windows as the more recent time windows are represented in multiple attributes. It also controls the number of attributes since we only have $\lceil \log_2(\mathtt{M}) \rceil + 1$ attributes for a given information window of $\mathtt{M}$ time windows instead of $\mathtt{M}$ attributes. With a time window of 0.5s and an information window of 20s, we have only 7 attributes referring to the average of previous 1, 2, 4, 8, 16, 32, 64 time windows. In PROTEUS, the time window size is 0.5s and the information window size is 20s by default without particular specification.

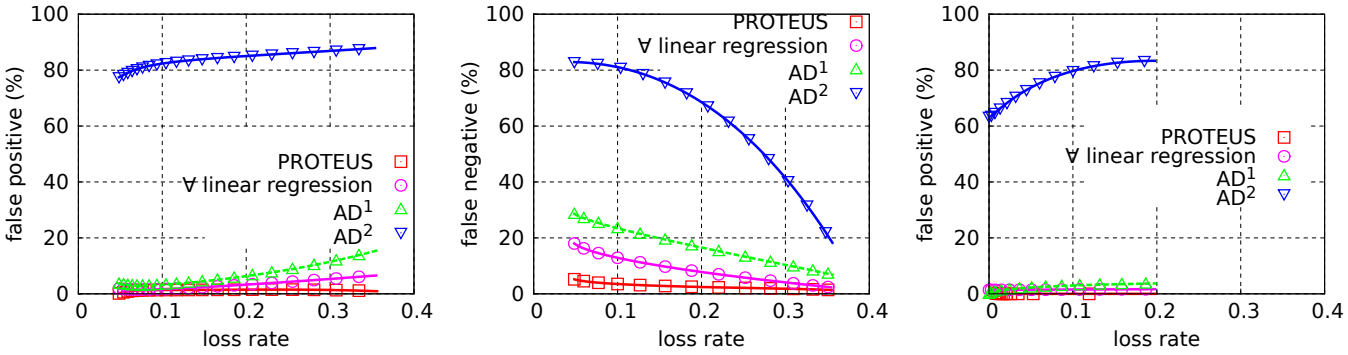
According to Figure 2(a), the correlation between the current network performance and the performance many seconds ago is too weak to be useful. Thus, PROTEUS does not perform offline training which can be inefficient and expensive. PROTEUS keeps feeding the regression trees with the most recent performance metrics, allowing the regression trees to perform hyper-correction internally. Based on the knowledge of performance histories, the regression trees output the respective performance predictions for the next time window.

4.3 Evaluating Forecast Accuracy

To evaluate the performance of PROTEUS, we carry out controlled experiments to measure its forecasting accuracy. The forecasting accuracy may be variable due to several aspects. When a mobile device is in communication over cellular networks, the network, the device, and even the different cellular technology generations under the same network can influence network predictability. Thus, we conduct the experiments using diverse setups as shown in Table 2. Table 2 lists the device, the network, and the cellular technology used. As TCP has built-in congestion control and retransmission, the objective of evaluating TCP is its direct impact to RTC applications, described in §5. To minimize bias, we repeat our experiments at different times in the day and at different locations. To directly control the sending rate and to easily observe loss, one-way delay, and throughput, we conduct most of our experiments over UDP. Even though most RTC applications operate directly over UDP, the applicability of PROTEUS is not limited by the transport protocol used by the application.

In the experiments, the mobile device communicates with a remote server in our campus in either the downlink or uplink direction and keeps the packet rate constant over the entire flow. To choose the sending rates, we saturate the end-to-end connection using UDP and estimate the bandwidth from the receiving rate. Based on the estimated bandwidth, the sending rate steps from 50% of the estimated bandwidth to 150% by 10% every time and each flow of a sending rate lasts at least one hour. On both the mobile and the remote server, we monitor packet traces using `tcpdump`. From the collected packet traces, we identify throughput, loss rate, and one-way delay. Here, we investigate one-way delay instead of round trip time because one-way delay is a more accurate measurement of network characteristics in a single direction and directly affects application performance. As mentioned in §3, PROTEUS performs clock drift compensation to compute one-way delay.

In the evaluation, we compare PROTEUS with general linear regression based prediction, representing the majority of on-demand prediction solutions [1, 2, 5, 7, 11]. Any given linear regression based prediction solution can be generalized as $\mathtt{P}[\mathtt{t}] = \lambda_1 \mathtt{P}[\mathtt{t}-1] + \lambda_2 \mathtt{P}[\mathtt{t}-2] + \cdots + \lambda_k \mathtt{P}[\mathtt{t}-\mathtt{k}]$, in which $\mathtt{P}[\mathtt{t}]$ is the network performance at time window $\mathtt{t}$ and $\lambda_{1,\cdots,\mathtt{k}}$ are the linear regression coefficients. However, deter-



(a) downlink false positive. (b) downlink false negative. (c) uplink false positive.

Figure 6: The accuracy in loss prediction. The X-axis shows the loss rate of each flow in FlowSet. The loss rate can be high because UDP's sending rate is agnostic of highly variable cellular links, which could commonly occur for UDP based applications.

mining the best $\lambda_{1,\cdots,k}$ for real-time use is challenging due to the diversity of network condition. To compare with linear regression based prediction, we perform linear regression offline for each experiment to obtain the $\lambda_{1,\cdots,k}$ that can best fit performance estimation, and hypothetically assume that the best $\lambda_{1,\cdots,k}$ is known a priori, which is an upper bound for any linear regression based solution. Besides, a common walkaround in engineering (named as AD^1 and AD^2) is to use the most recent network performance as follows.

- AD^1 assumes the performance in the next time window is the same as the performance observed in the current time window.
- AD^2 is less aggressive than AD^1 to circumvent random variation in cellular networks. AD^2 assumes that the performance in the next time window is the average of the performance in the information window, i.e., the same as PROTEUS's information window.

Although AD^1 and AD^2 may seem simple and naïve, most existing RTC applications estimate loss rate and one-way delay using such methods since they work relatively well on wired networks where statistics are relatively stable.

PROTEUS's performance in loss prediction. We first investigate PROTEUS's accuracy in predicting the presence of any packet loss in the next time window by measuring the false positive and false negative rates in FlowSet. A false positive occurs when PROTEUS predicts a packet loss in the time window but there is none. False positives can cause inefficient usage of network resources. For example, an application may introduce unnecessary FEC, which would reduce the rate available for the actual coding of media. In contrast, a false negative occurs when PROTEUS predicts no packet loss but a packet is lost in a time window. For many applications, the false negatives may result in degraded application performance due to unrecoverable packet loss. This is especially true for RTC applications where retransmissions are not possible.

In Figures 6(a)-(c), we show the false positive and false negative rates as a function of the actual loss rate for downlink and uplink traffic. From Figure 6(a), we see that PROTEUS's false positive rate is consistently around <1% when the loss rate is in the range from 0.05 to 0.4, while AD^1's false positive rate is around 2-5% when the loss rate is less than 0.3 and increases to 20% after the loss rate grows to 0.35. AD^2 performs even worse with the false positive rates higher than 80%. The hypothetical upper bound for any linear regression based prediction ("∀ linear regression") is around 3%. Note that the loss rate can be as high as 0.35, because UDP's sending rate is agnostic of highly variable cellular network performance [14, 45], which could commonly occur for UDP based applications. There are two reasons that PROTEUS outperforms linear regression techniques: (i) PROTEUS uses cross statistics, e.g., the use of loss in delay prediction and vice versa; and (ii) regression trees have some amount of non-linearity. Consequenly, without fully capturing the performance trend in previous time windows, AD^1 and AD^2 can only achieve the accuracy as poor as random guessing by merely estimating based on the last time window or aggregated history information.

In Figure 6(b), we show the false negative rate of the flows in FlowSet. As mentioned, a false negative is more critical than a false positive, for RTC applications. From Figure 6(b), PROTEUS's false negative rate is consistently around 1-3%, AD^1's false negative is 5-30%, and AD^2's false negative is around 20-80%. The upper bound for linear regression based approaches is 5-20%, which is much worse than PROTEUS. When the loss rate is low, the running flow may not provide sufficient information of true positives, i.e., lost packets. Thus, as the loss rate decreases, PROTEUS, AD^1, and AD^2 all have increasing false negative rates. Although AD^2's false negative rate is only slightly worse than PROTEUS, PROTEUS's false positive rate is significantly better. Compared to AD^1, AD^2 has low false positive rate, but does not retain a low false negative rate at the same time.

Besides downlink traffic, we also investigate uplink traffic as uplink traffic is significant in applications such as Skype, FaceTime, and Google+ Hangouts. Similar to Figure 6(a), PROTEUS is slightly better than the hypothetical upper bound of linear regression solutions and AD^1, and much better than AD^2 in false positive rate according to Figure 6(c). PROTEUS's false positive rate is less than 2% over the flows in FlowSet, "∀ linear regression"'s and AD^1's false positive rates range from 1% to 3%, and AD^2's false positive is no less than 60%. Due to space limitation, we only include the false positive for uplink flows, but we observe similar perfor-

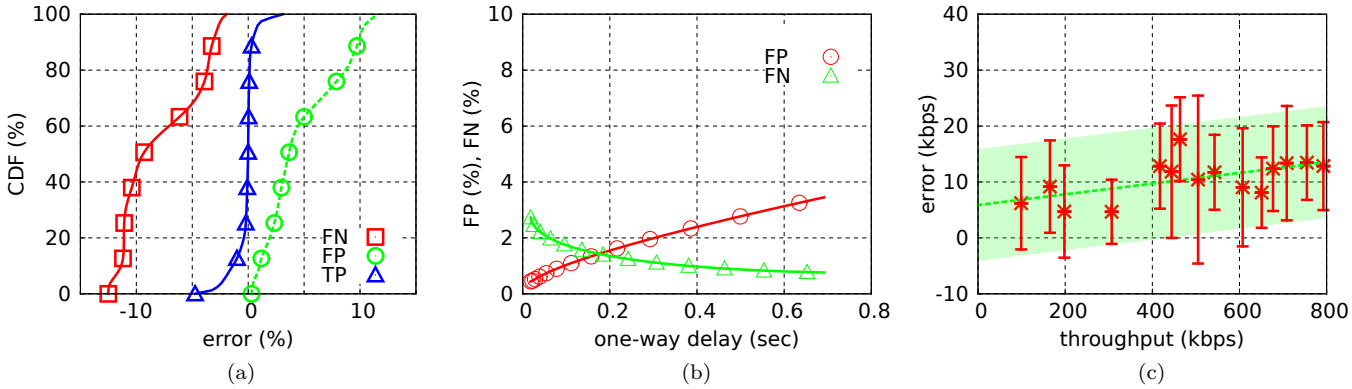


Figure 7: The quantitative accuracy on (a) loss rate, (b) one-way delay, and (c) throughput. In (c), only 15 randomly chosen flows are presented to avoid overlapping.

mance for PROTEUS, AD[1], and AD[2] in their false negative for uplink flows.

The false positive and false negative results for the packet loss presented above show the prediction accuracy. Moreover, the regression tree used in PROTEUS also predicts the packet loss rate quantitatively rather than giving a simple binary decision of whether there will be packet loss. In Figure 7(a), we show the accuracy of this quantitative prediction of packet loss rate. We show the error in packet loss rate prediction for each of the three categories according to the error type, i.e., false negative, false positive, and true positive. Only true negatives do not lead to error because both the predicted and the ground truth loss rate are zero. We show the cumulative distribution function (CDF) of the error for each of these three error types in Figure 7(a).

Among true positive predictions, PROTEUS could either overestimate or underestimate the loss rate. If the loss rate is overestimated, similar to false positives, applications may introduce unnecessary FEC. If loss rate is underestimated, similar to false negatives, applications may experience information loss, which may be more critical. In Figure 7(a), 70% of true positive predictions have no error or the negligible error of less than 0.1% and 99% of them have the error within 5%. PROTEUS's median false negative error is around 10%. With false positives, PROTEUS's median error is 2% and 95% of predictions are less than 10%, which indicates that our design will very unlikely make applications introduce unnecessary protection traffic.

PROTEUS's performance in delay prediction. Besides loss rate prediction, RTC applications can also benefit from one-way delay prediction. One-way delay prediction allows applications such as video conferencing and VoIP to appropriately set their de-jitter buffer sizes, which determines how long the application waits for a packet before declaring it to be lost. Most VoIP and video conferencing systems do not use a fixed de-jitter buffer. The de-jitter buffer starts at something small and grows to some maximum value if packets do not arrive in time. This maximum is set to something which is tolerable for the given application. For example, a 150-200ms end-to-end delay is the maximum that video conferencing systems can tolerate [16]. Once packets start to arrive in time, the de-jitter buffer size shrinks according to some schedule (e.g., 10% every 5s) to lower the end-to-end delay seen by the application. Usually, de-jitter buffer size adaptation is accomplished by using an adaptive rate playout (either speeding up or slowing down media playback) [15]. Accurately predicting one-way delay can allow an application to more intelligently control this adaptation to improve performance.

Inappropriate de-jitter buffer size selection can result in suboptimal performance. If it is set larger than needed to absorb the inherent network delay variation, the application suffers from larger delay than needed. Otherwise, if it is set too small, a large number of packets that arrive late will be declared lost which results in poor application performance. In Figure 7(b), we show the accuracy of PROTEUS in predicting if the delay in the following time window will exceed the tolerable end-to-end delay (i.e., 150ms [16]) as a function of the true one-way delay. This can answer the question whether we should grow the de-jitter buffer size. Note that the one-way delay is calibrated so that the one-way delay is 0ms if there is no congestion on the network [35]. Similar to before, a false positive occurs when we predict the delay to be larger than 150ms but in reality it is not. A false negative is when we predict the delay to be smaller than 150ms, but in reality it is larger. According to Figure 7(b), the false positive is around $<3\%$ and the false negative is around $<2\%$.

PROTEUS's performance in throughput prediction. Predicting the throughput in the next time window is also important for certain bandwidth intensive applications such as video conferencing and VOD. For RTC applications such as video conferencing, throughput prediction can be combined with loss rate prediction to appropriately set the encoder bitrate and the FEC rate. For non-RTC applications such as VOD using chunked video streaming, throughput prediction can be used to appropriately assign bit allocation across video chunks to improve video quality. Overestimating the achievable throughput may lead to application-level information loss or increased delay, while underestimating the achievable throughput prevents applications from maximally utilizing the network.

Since PROTEUS passively monitors network traffic, it cannot predict the achievable throughput of the following time window at an arbitrary sending rate. PROTEUS makes the prediction assuming the sending rate of the next time window is the same as the previous one. We evaluate PROTEUS's accuracy in throughput prediction as a function of

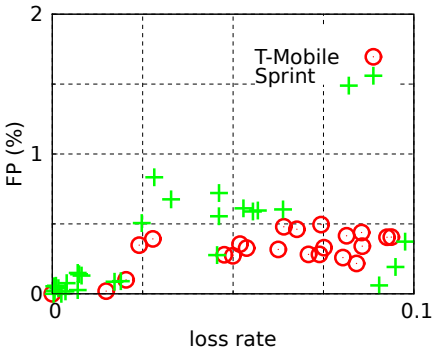


Figure 8: The impact of the cellular network.

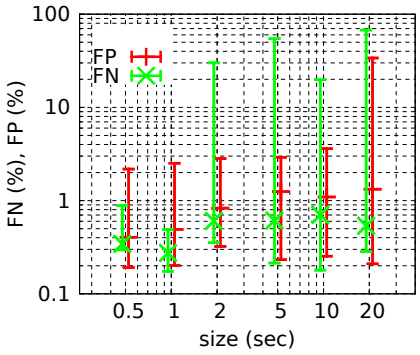


Figure 9: The impact of the time window size.

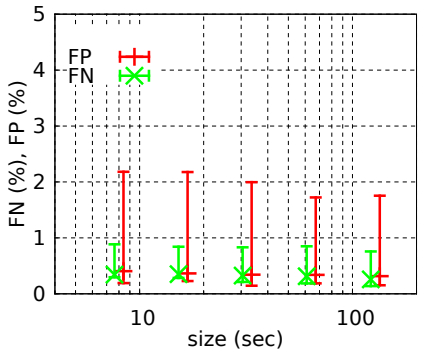


Figure 10: The impact of the information window size.

the actual throughput and show the results in Figure 7(c). To ensure Figure 7(c)'s readability, 15 flows in FlowSet are randomly chosen. Each flow corresponds to an error bar showing the median and standard deviation of the predicted throughput over all the time windows. We see that throughput prediction is fairly accurate across the entire range of throughput, i.e., the median error is around 10kbps with a standard deviation of around 10kbps.

PROTEUS's performance in other networks. So far, we have investigated the accuracy of using PROTEUS to predict the loss rate, one-way delay, and throughput for one particular cellular carrier. To be broadly applicable, we need to verify if PROTEUS's accuracy applies to other cellular carriers as well. Figure 8 shows the false positive rate of loss predictions for T-Mobile and Sprint, whose setup is described in Table 2. According to Figure 8, both T-Mobile and Sprint have even a lower false positive than AT&T. Due to space limitation, we only include the evaluation on false positive rate of T-Mobile and Sprint, but have verified that PROTEUS also has high prediction accuracy in terms of the false negative and false positive rates in one-way delay and throughput prediction.

PROTEUS's performance affected by the time and information window size. We also study the impact of the time window size and the information window size. We have investigated the impact of the time window size on autocorrelation coefficient in Figure 2(a). Here we investigate the impact of time window size on PROTEUS's prediction accuracy. Figure 9 shows the distribution of the false positive and false negative rates as functions of the time window size. Unlike Figure 2(a), PROTEUS's false positive rate is the minimum when the time window size is 0.5s, while PROTEUS's false negative is the minimum when the time window size is 1s. If the time window size further increases, the variation of the false positive can be extremely high (>50%). Using a large time window with a fixed information window size suffers from high performance variability due to the fact that only a few time windows are available. Using a small time window with the same amount of history allows predicting performance on a finer granularity and is preferred. We choose a time window of 0.5s as the lower bound since it is reasonable to keep the time window larger than typical network round trip time.

Another parameter is PROTEUS's information window size which in conjunction with the time window size determines the number of time windows used in the prediction. Similar to the autocorrelation results in Figure 2(b), we see from Figure 10 that prediction gains from larger information windows in PROTEUS are marginal. Thus we can achieve accurate prediction using time windows within the last 8s.

5. APPLICATION OF PROTEUS

Applications can take advantage of network metric prediction to improve their performance. The exact benefit will depend on how each application uses the information obtained from PROTEUS. Here we show the improvement PROTEUS can provide to two RTC applications: a video conferencing application and an interactive software application, i.e., Draw Something.

5.1 Video Conferencing System

Equivalent video conferencing emulation. We describe how we realistically evaluate the video conferencing application to make use of PROTEUS in a working system, considering all the overhead of the associated changes with the small difference of running the client software on a legacy laptop instead of a mobile device due to software constraints.

There are standard methods to evaluate video conferencing using the H.264/AVC reference software [4] on PCs. However, there is no such equivalent open-source encoding/decoding suite on mobile platforms. Thus, to best effectively evaluate video conferencing on mobile platforms, we take the following steps: (i) we modify the H.264/AVC reference software so that the H.264 encoder can adjust the video codec bitrate and FEC rate for each video frame rather than sticking to fixed codec bitrate and FEC rate for an entire video stream; (ii) we use a 2GHz dual core Thinkpad T60 with 2G RAM to run the H.264 decoder, with the computation performance as close to off-the-shelf mobile devices as possible; and (iii) we replay the packet traces collected in cellular networks (i.e., FlowSet) to emulate mobile network performance for H.264, which ensures identical, reproducible, and representative mobile network conditions.

The packet sequencing and timing information seen by PROTEUS is exactly the same as it is in FlowSet, enabling PROTEUS to make the same predictions as it does in cellular networks. The emulation setup results in only two overestimations in evaluating the perceptual video quality. One is due to the Thinkpad T60 being more powerful than a mobile device, and the other is due to the elimination of the

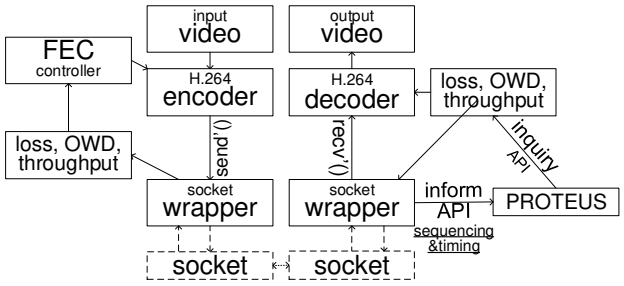


Figure 11: The platform setup to emulate mobile video conferencing. The dashed blocks and arrows are replayed using FlowSet to guarantee realistic mobile network condition.

cost for traffic transmitting over its network interface. The overestimations should be acceptable given that video conferencing applications such as Skype can work perfectly well on mobile devices with excellent network connections.

As depicted in Figure 11, we focus on evaluating the performance in downlink video conferencing, i.e., the video stream is adaptively encoded and sent from the server (either a content server or the other mobile device in P2P scenarios) to the client, where we can evaluate the perceptual quality of the received video stream. The H.264 decoder in Figure 11 represents the video conferencing application on the client, i.e., the mobile device emulated by the 2GHz dual core Thinkpad T60, while the H.264 encoder and FEC controller represent the counterpart on the server.

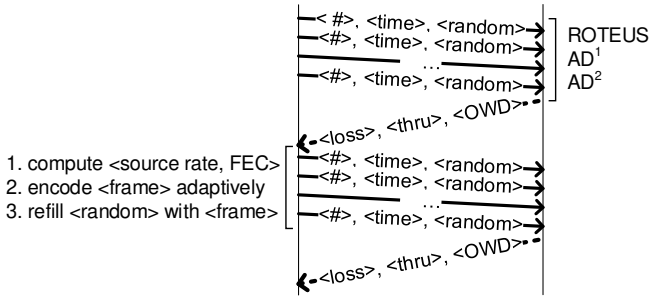


Figure 12: The packet trace replaying to guarantee reproducible mobile network conditions.

As shown in Figure 12, along a packet trace in FlowSet, each UDP packet from the sender includes a sequence number, a timestamp, and some random content. The receiver sends a placeholder packet at the end of every time window. When the packet trace is being collected, those placeholder packets are filled with random values. However, we refill these packets with the predicted performance metrics later in evaluating PROTEUS, AD^1, or AD^2. Once the sender sees a placeholder packet refilled with performance predictions, it computes the best source coding rate and FEC, and encodes the next frames accordingly. After the encoding, it refills the random content in UDP packets with newly encoded frames. Eventually, we decode the replayed packet trace on the receiver side and evaluate the perceptual quality.

By replaying network traces collected from the cellular network in FlowSet, we can reproduce representative mobile network conditions. We measure the video quality performance using standard video sequences shown in Table 3 (referred as VideoSet) [9]. As illustrated in Figure 11, given a video sequence in VideoSet, the H.264 encoder on the server translates the video sequence to a RTP network stream. To test the video quality under various and reproducible network conditions, for each flow in FlowSet, we replace its content with the encoded network stream respectively. On the client, the H.264 decoder decodes the received network stream and constructs the video for users. According to FlowSet, if a packet is lost or does not arrive in time at the de-jitter buffer, the corresponding frame may be corrupted so that the perceptual quality of the decoded video will be affected.

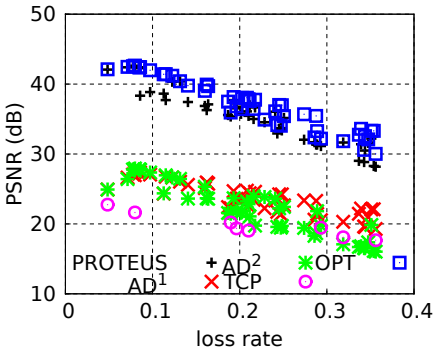
PROTEUS utilization in video conferencing. Based on past work, there are well established adaptations that applications take in response to changing network conditions. The quality of video conferencing is very sensitive to three network metrics, i.e., the packet loss, the one-way delay, and the throughput. To ensure good quality, the video conferencing application can tune several parameters.

- The video codec can adjust the compression ratio, trading source video quality with codec bitrate, by adjusting the quantization parameter (QP) of macroblocks [29, 30]. A smaller QP value results in better visual quality at the expense of a higher bitrate and higher bandwidth requirement.
- The application can insert an appropriate number of forward error correction (FEC) packets. This process allocates the total transmission rate amongst source bitrate and channel (error correction) bitrate.
- The application can adjust the de-jitter buffer size, which specifies how long the receiving end waits for data to arrive before it is played back. The de-jitter buffer size trades application latency with *late loss*. A larger de-jitter buffer size increases the chance that a packet will arrive in time for playback, but increases the end-to-end application latency. For video conferencing applications, the user cannot perceive an end-to-end delay (including propagation delay, network queuing delay, and time spent in the de-jitter buffer) lower than 200ms [10]. However, when the end-to-end delay is above this threshold, there is perceived degradation.

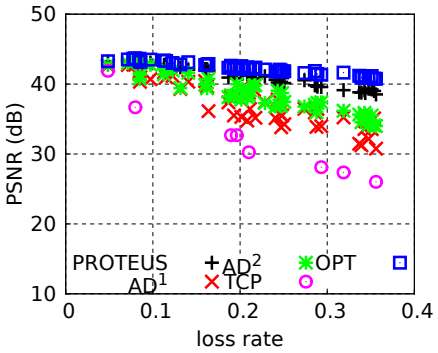
video	# of frames	sequence	resolution	size (MB)
akiyo	300	10/IPPP	QCIF,CIF	44
bowing	300	10/IPPP	QCIF,CIF	44
bridge(close)	2001	10/IPPP	QCIF,CIF	291
bridge(far)	2101	10/IPPP	CIF	305
foreman	300	10/IPPP	QCIF,CIF	44
highway	2000	10/IPPP	CIF	291

Table 3: The video sequences to reproduce video conferencing streams (VideoSet). The frame rate for all these video sequences is 30fps.

We denote the predicted packet loss rate, one-way delay, and throughput by the triplet with $(\epsilon, \delta, \mathtt{T})$. This is the predicted network performance obtained from the regression tree on the mobile device and delivered to the server. Given this network prediction, we set the application parameters as follows: (i) the video codec bitrate for a given frame is set to $\mathtt{R} = (1 - \alpha\epsilon)\mathtt{T}$ $(\alpha \geq 1)$; (ii) the FEC rate to $\alpha\epsilon\mathtt{T}$; and (iii) the receiver de-jitter buffer size to $\beta\delta$ seconds. α is a constant that determines the actual amount of FEC protec-



(a) 5th-percentile PSNR.



(b) 25th-percentile PSNR.

Figure 13: The PSNR for PROTEUS, AD^1, AD^2, TCP, and OPT, which assumes that we a priori know the exact loss rate.

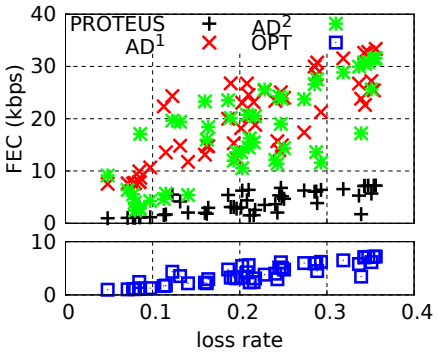


Figure 14: The FEC overhead. "OPT" shows the total FEC overhead in the optimal scenarios.

tion used. In the evaluation, $\alpha = 1$. The total estimated throughput $\mathtt{T}$ is divided amongst source rate and FEC rate. As an example, suppose $\mathtt{T}$ is estimated as 500kbps, the estimated loss rate is $\epsilon = 0.06$, and $\alpha = 2$ is used. With this, 440kbps should be used for source coding and 60kbps for FEC protection. With a video frame rate of 15fps and a packet size of 500 bytes, this approximately translates to 7 packets per frame of source coding and 1 packet per frame of FEC.

Once the video frames are coded into packets, the evaluation platform assumes that these packets go through a network that has characteristics obtained from the collected network trace. If the actual loss on the network is less than or equal to $\alpha\epsilon\mathtt{T}$, the receiver has sufficient information to decode the frame, otherwise the frame is assumed to be lost. In addition to packet loss on the network, we also consider delay in the evaluation, assuming a de-jitter buffer size of $\beta\delta$ ($\beta \geq 1$), which is slightly larger than the estimated δ. In the above example, if two or more packets are lost or delayed later than $\beta\delta$, the respective frame is lost. In the evaluation, $\beta = 1$.

Perceptual video quality assessment. The H.264 decoder provides the ability to conceal lost frames. Upon decoding the sequence, which may potentially have some frames lost, we compare the decoded video sequence to the original and measure the peak signal-to-noise ratio (PSNR), which is a commonly used metric to measure video quality. This setup is used to repeatedly measure the video quality using the prediction obtained from PROTEUS, AD^1, and AD^2 over the various video sequences in VideoSet and over the various network traces in FlowSet.

If we underestimate ϵ (i.e., loss rate), then the application may add insufficient redundancy, which results in unrecoverable packet loss for the application. Otherwise, if we overestimate ϵ, then the application may add too much redundancy causing the video codec bitrate to be lower than needed, resulting in poorer video quality. Thus, both underestimation and overestimation of the future network performance can lower the perceptual video quality. We analyze the effect of network parameter mis-prediction on video quality for the three prediction techniques, PROTEUS, AD^1, and AD^2. We also analyze the performance hypothesizing that we exactly know the loss rate and insert just the right amount of FEC to counteract any packet loss. It is possible as we have the network traces. This optimal scheme is denoted as OPT.

For each video sequence in VideoSet, we run all the network traces from FlowSet using each of the four methods (PROTEUS, AD^1, AD^2, and OPT). For each of the four methods, we compute a distribution of PSNR. In Figure 13, we show the 5th and 25th percentiles of the PSNR as a function of loss rate for each of the four methods. We purposefully show the low PSNR values in the distribution since this is what most affects overall user satisfaction and is a measure of how each of the three schemes performs in tough network conditions. In addition to the four adaptive methods using UDP, we evaluate the adaptive bitrate streaming over TCP as well assuming that the encoded bitrate is always adapted to the throughput in the current time window. To guarantee the network conditions for TCP and UDP experiments are comparable, for each TCP flow, we create UDP packets back-to-back lasting for the same duration. In Figure 13, the distribution of PSNR for TCP flows is represented as the function of the loss rate of the back-to-back UDP flow's loss rate.

In general, if a frame is lost, the PSNR of the other frames that reference the lost frame becomes very low, e.g., PSNR <25dB. Since PROTEUS can accurately predict loss rate with almost 99% accuracy, there are very few cases (i.e., <1%) where we see unrecoverable packet loss. Thus, as we see from Figure 13(a), even the 5th percentile PSNR shows no packet loss using PROTEUS. The only reason the 5th percentile PSNR is below OPT is due to over-protection, which reduces the video codec bitrate. This reduction in PSNR in PROTEUS when compared to OPT (due to over-protection) is on the order of 1dB, whereas the reduction in PSNR in AD^1 and AD^2 (due to unrecoverable lost packets) is on the order of 15dB. As loss rate increases, the PSNR using PROTEUS decreases but stays close to OPT. It is obvious that increasing loss rate results in decreased capacity. Since a larger portion of the total throughput has to be allocated to FEC, we see a reduction in PSNR. To show the visual effect of unrecoverable packet loss, we show representative frames in the 5th percentile of PSNR values in Figure 15 using the schemes AD^{1,2}, TCP, and PROTEUS. It is clear that unrecoverable packet loss results in substantial video quality degradation. As PROTEUS may overestimate the actual loss rate, there will inevitably be some cases where

(a) 36dB – PROTEUS. (b) 23dB – AD^1. (c) 23dB – AD^2. (d) 20dB – TCP.

Figure 15: The perceptual video quality corresponding to the 5th-%ile PSNR (using "akiyo" in Table 3).

we over-protect. In Figure 14, we show the bitrate taken by FEC packets which are not used to correct any loss. The bottom plot shows the amount of needed FEC to correct for packet loss. As we see, as the loss rate increases, the amount of needed protection also increases. The top plot in Figure 14 shows the amount of additional FEC beyond the needed amount which is applied when using each of the three schemes to predict the loss rate. We see that indeed PROTEUS does add a certain amount of FEC overhead. However, it is fairly small, and as we have seen from Figure 13, it only results in about a 1dB reduction in PSNR. Even at high loss rates, this overhead is less than 10kbps. The overhead for AD^1 and AD^2 is actually much higher as it only reacts once loss has already started occurring. Just because there was loss in the previous time window does not guarantee that there will be loss in the future time windows. So not only do AD^1 and AD^2 suffer from unrecoverable packet loss, they also actually insert more FEC than PROTEUS.

5.2 Interactive Software Application

Many interactive software applications have bursty traffic patterns, e.g., the burst can consist of a screen update. A good measure of the performance of such interactive software applications can be the time taken to deliver the burst of data. However, as opposed to throughput sensitive applications such as file downloading, a burst is relatively small in duration and thus the latency caused by packet loss recovery becomes significant in determining performance. Therefore, an adaptive UDP based scheme that appropriately controls the throughput allocated to source and FEC becomes important in determining application performance [33], especially for the channels with packet loss such as wireless networks.

Here we consider a particular application, Draw Something, to show how PROTEUS can benefit interactive software applications such as desktop sharing and multi-player gaming. In Draw Something, two players alternate turns between drawing a picture to convey a guessword for the partner to guess. In each turn, the stroke-by-stroke drawing animation playing back to the partner results in a traffic burst. We rely on trace analysis to evaluate how PROTEUS can benefit such an application. We capture the application traffic via `tcpdump` on Android and identify the traffic bursts corresponding to drawing animations. The trace consists of a duration lasting five hours and consisting of hundreds of bursts.

To measure the improvement of an adaptive UDP scheme using PROTEUS over TCP, we replay the traffic using UDP plus PROTEUS (UDP/PROTEUS) and measure the time between the first packet of a burst being sent to the last packet in the burst being received. UDP/PROTEUS adds an appropriate amount of FEC if packet loss is anticipated. A poor prediction of loss rate can result in higher burst delivery time due to either retransmission or over-protection. To verify that the traffic replaying under UDP/PROTEUS experiences roughly the same network conditions as the Draw Something's original trace over TCP, we compare their median perceptual delay. Figure 16 shows the distribution of the perceptual delay over different burst size. As the median perceptual delay for UDP/PROTEUS is very close to TCP's, we expect that UDP/PROTEUS experiences similar network conditions. According to Figure 16, when the burst size is very small, TCP and UDP/PROTEUS perform equally at any percentile in the distribution. However, when the burst size grows to 10KB and larger (e.g., 20KB), UDP/PROTEUS performs much better, particularly in the 75-90th percentile perceptual delay, which may easily produce user dissatisfaction. UDP/PROTEUS reduces the 90th percentile perceptual delay by roughly 1s when the burst size is around 10KB. When the burst size is around 20KB, UDP/PROTEUS's 90th percentile perceptual delay is 2s while TCP's is up to 6s.

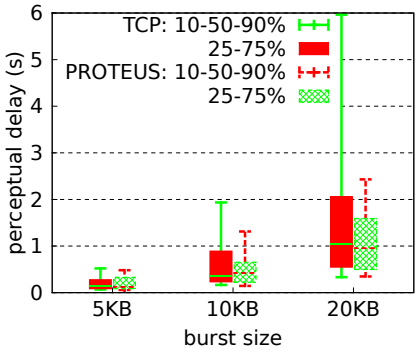


Figure 16: The perceptual delay in Draw Something.

6. RELATED WORK

There has been a significant amount of previous work proposing solutions for adapting to varying network performance [1, 2, 5, 7, 19, 22, 23, 34, 42, 48]. However, these proposed solutions adjust the application behavior based on the "current" network condition. By then, performance degradation may be observed already. PROTEUS is fundamentally different from these previous adaptation approaches in that PROTEUS proactively forecasts the achievable network performance rather than passively reacting to performance

degradation. This allows applications to modify their behavior in anticipation of degrading network condition which can significantly improve performance.

The key to the effectiveness of PROTEUS is network performance predictability. There have been previous measurement studies to investigate the network behavior of cellular networks [13, 19, 20, 24, 31, 32, 44]. Among these measurement studies, the existence of network predictability has been indicated from certain aspects. For example, Liu et al. observed that the time for a mobile device to stay in any particular data rate state is on the order of hundreds of time slots (with each time slot being 1.67ms) in a EV-DO network [31]. Manweiler et al. found that in AT&T and T-Mobile networks, the network latency measured in a 15-minute time window has certain dependency on the latency of a previous time window [32]. Standing upon the shoulder of previous studies, PROTEUS answers the following three questions. First, how predictable are cellular networks? Second, what network features can be used for performance prediction? Third, how accurate is the prediction?

There has also been a significant amount of previous work that studies predictability in mobile (cellular) networks with the goal of improving application performance [32, 37, 39, 40, 46]. Access pattern predictability is investigated with the attempt to reduce connectivity overhead due to frequent user mobility in [36, 37] and with the attempt to switch to better access networks in [12, 38, 43]. PROTEUS distinguishes itself from these studies in two aspects. First, PROTEUS forecasts network performance at run time, rather than offline. As cellular networks are well-known to be highly variable, a one-time, offline prediction will not work well because it will be quickly out-dated. Second, PROTEUS's performance prediction is much more fine-grained. Unlike previous studies that usually attempt to improve performance for non-real-time bandwidth intensive applications, which are not sensitive to packet loss and one-way delay, PROTEUS targets a broad spectrum of applications including RTC applications that are sensitive to packet loss and one-way delay.

To the best of our knowledge, our study is the first one proposing a fine-grained, real-time solution for predicting future network performance over cellular networks, which allows applications to react before network degradation occurs.

7. CONCLUDING REMARKS

In this study, we systematically quantified the predictability of network performance metrics in three major cellular networks: AT&T, T-Mobile, and Sprint. For all three carriers, we identified the existence of strong predictability of network performance, even over a short term window e.g., 0.5s. To take advantage of this predictability, we proposed a system PROTEUS, which collects network performance information being observed by applications, employs regression trees to learn network performance patterns, and forecasts future network performance to benefit application performance. PROTEUS can predict the occurrence of packet loss within a time window for 98% of the time windows, occurrence of long one-way delay for 97% of the time windows, and the throughput within a median error of 10kbps with average throughput ranging from 100–800kbps. By using PROTEUS in a video conferencing scenarios, we can improve the PSNR of the perceived video by 15dB when compared against traditional performance adaptation techniques. Compared with the hypothesized optimal scenario in which the application knows exactly which packets will be lost, PROTEUS is only 1–2dB worse.

We believe that PROTEUS is a comprehensive and effective network performance forecasting framework for use by mobile applications especially for use over cellular networks. PROTEUS can also inspire the development of challenging applications on mobile devices.

8. ACKNOWLEDGMENTS

We would like to thank the reviewers and Koen Langendoen for shepherding the paper. This research was sponsored in part by NSF grants CNS-1059372, CNS-1050157, CNS-1039657, and CNS-0939707 and by Navy award N00014-09-1-0705. .

9. REFERENCES

[1] Adobe HTTP Dynamic Streaming. http://www.adobe.com/products/hds-dynamic-streaming.html.

[2] Apple HTTP Live Streaming. https://developer.apple.com/resources/http-streaming.

[3] Cisco TelePresence Secure Communications and Signaling. http://www.cisco.com/en/US/docs/solutions/Enterprise/Video/telepresence.html.

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

[5] IIS Smooth Streaming. http://www.iis.net/download/SmoothStreaming.

[6] ISO/IEC DIS 23009-1.2 Dynamic Adaptive Streaming over HTTP (DASH). http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57623.

[7] Octoshape Broadcasting Flash Media Multi-Bit Rate. https://support.octoshape.com/entries/20655458-broadcasting-flash-media-multi-bit-rate.

[8] RTCWeb Status Pages. http://tools.ietf.org/wg/rtcweb/.

[9] YUV Video Sequences. http://trace.eas.asu.edu/yuv.

[10] ITU-T recommendation G. 114, 2000.

[11] S. Akhshabi, A. Begen, and C. Dovrolis. An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP. In *Proc. ACM MMSys*, 2011.

[12] A. Aljadhai and T. Znati. Predictive Mobility Support for QoS Provisioning in Mobile Wireless Environments. *IEEE J. Selected Areas in Communications*, 2001.

[13] N. Antunes, C. Fricker, P. Robert, and D. Tibi. Metastability of CDMA Cellular Systems. In *Proc. ACM MOBICOM*, 2006.

[14] A. Balasubramanian, R. Mahajan, and A. Venkataramani. Augmenting Mobile 3G Using WiFi. In *Proc. ACM MobiSys*, 2010.

[15] J.-C. Bolot and A. Vega-García. Control Mechanisms for Packet Audio in the Internet. In *Proc. IEEE INFOCOM*, 1996.

[16] A. Bouch, M. Sasse, and H. DeMeer. Of Packets and People: A User-Centered Approach to Quality of Service. In *Proc. IEEE IWQoS*, 2000.

[17] L. Breiman. *Classification and Regression Trees*. Chapman & Hall/CRC, 1984.

[18] T. Bu, L. Li, and R. Ramjee. Generalized Proportional Fair Scheduling in Third Generation Wireless Data Networks. In *Proc. IEEE INFOCOM*, 2006.

[19] R. Chakravorty, S. Banerjee, P. Rodriguez, J. Chesterfield, and I. Pratt. Performance Optimizations for Wireless Wide-Area Networks: Comparative Study and Experimental Evaluation. In *Proc. ACM MOBICOM*, 2004.

[20] M. Chan and R. Ramjee. TCP/IP Performance over 3G Wireless Links with Rate and Delay Variation. In *Proc. ACM MOBICOM*, 2002.

[21] M. Chuah, W. Luo, and X. Zhang. Impacts of Inactivity Timer Values on UMTS System Capacity. In *IEEE Conference on Wireless Communications and Networking Conference*, 2002.

[22] I. Haratcherev, J. Taal, K. Langendoen, R. Lagendijk, and H. Sips. Optimized Video-Streaming over 802.11 by Cross-Layer Signalling. *IEEE Computer Magazine*, 44(1):115–121, 2006.

[23] B. Higgins, J. Flinn, T. Giuli, B. Noble, C. Peplin, and D. Watson. Informed Mobile Prefetching. In *Proc. ACM MobiSys*, 2012.

[24] J. Huang, Q. Xu, B. Tiwana, Z. Mao, M. Zhang, and P. Bahl. Anatomizing Application Performance Differences on Smartphones. In *Proc. ACM MobiSys*, 2010.

[25] A. Jalali, R. Padovani, and R. Pankaj. Data Throughput of CDMA-HDR a High Efficiency-High Data Rate Personal Communication Wireless System. In *Proc. IEEE VTC*, 2000.

[26] H. Jiang, Z. Liu, Y. Wang, K. Lee, and I. Rhee. Understanding Bufferbloat in Cellular Networks. In *Proc. ACM SIGCOMM IMC*, 2012.

[27] N. Kolehmainen, J. Puttonen, P. Kela, T. Ristaniemi, T. Henttonen, and M. Moisio. Channel Quality Indication Reporting Schemes for UTRAN Long Term Evolution Downlink. In *Proc. IEEE VTC*, 2008.

[28] H. Kushner and P. Whiting. Convergence of Proportional-Fair Sharing Algorithms under General Conditions. *IEEE Trans. Wireless Communications*, 2004.

[29] H. Lee, T. Chiang, and Y. Zhang. Scalable Rate Control for MPEG-4 Video. *IEEE Trans. Circuits and Systems for Video Technology*, 2000.

[30] Z. Li, F. Pan, K. Lim, X. Lin, and S. Rahardja. Adaptive Rate Control for H. 264. In *Proc. IEEE ICIP*, 2004.

[31] X. Liu, A. Sridharan, S. Machiraju, M. Seshadri, and H. Zang. Experiences in a 3G Network: Interplay between the Wireless Channel and Applications. In *Proc. ACM MOBICOM*, 2008.

[32] J. Manweiler, S. Agarwal, M. Zhang, R. Roy Choudhury, and P. Bahl. Switchboard: A Matchmaking System for Multiplayer Mobile Games. In *Proc. ACM MobiSys*, 2011.

[33] S. Mehrotra, J. Li, and Y.-Z. Huang. Optimizing FEC Transmission Strategy for Minimizing Delay in Lossless Sequential Streaming. *IEEE Trans. Multimedia*, 2011.

[34] I. Mohomed, J. Cai, S. Chavoshi, and E. De Lara. Context-Aware Interactive Content Adaptation. In *Proc. ACM MobiSys*, 2006.

[35] S. Moon, P. Skelly, and D. Towsley. Estimation and Removal of Clock Skew from Network Delay Measurements. In *Proc. IEEE INFOCOM*, 1999.

[36] A. Nicholson, Y. Chawathe, M. Chen, B. Noble, and D. Wetherall. Improved Access Point Selection. In *Proc. ACM MobiSys*, 2006.

[37] A. Nicholson and B. Noble. Breadcrumbs: Forecasting Mobile Connectivity. In *Proc. ACM MOBICOM*, 2008.

[38] J. Pang, B. Greenstein, M. Kaminsky, D. McCoy, and S. Seshan. WiFi-Reports: Improving Wireless Network Selection with Collaboration. In *Proc. ACM MobiSys*, 2009.

[39] S. Schubert, F. Uyeda, N. Vasic, N. Cherukuri, and D. Kostic. Bandwidth Adaptation in Streaming Overlays. In *Proc. IEEE COMSNETS*, 2010.

[40] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V. Padmanabhan. Bartendr: A Practical Approach to Energy-Aware Cellular Data Scheduling. In *Proc. ACM MOBICOM*, 2010.

[41] H. Schulzrinne. RTP: A Transport Protocol for Real-Time Applications. *IETF, Request For Comments 3550*, 2003.

[42] S. Sen, N. Madabhushi, and S. Banerjee. Scalable WiFi Media Delivery through Adaptive Broadcasts. In *Proc. USENIX NSDI*, 2010.

[43] L. Song, U. Deshpande, U. Kozat, D. Kotz, and R. Jain. Predictability of WLAN Mobility and Its Effects on Bandwidth Provisioning. In *Proc. IEEE INFOCOM*, 2006.

[44] W. Tan, F. Lam, and W. Lau. An Empirical Study on 3G Network Capacity and Performance. In *Proc. IEEE INFOCOM*, 2007.

[45] F. Tso, J. Teng, W. Jia, and D. Xuan. Mobility: A Double-Edged Sword for HSPA Networks: A Large-Scale Test on Hong Kong Mobile HSPA Networks. In *Proc. ACM MobiHoc*, 2010.

[46] R. Wolski, N. Spring, and C. Peterson. Implementing A Performance Forecasting System for Metacomputing The Network Weather Service. In *Proc. IEEE Supercomputing*, 1997.

[47] Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman. Identifying Diverse Usage Behaviors of Smartphone Apps. In *Proc. ACM SIGCOMM IMC*, 2011.

[48] Z. Zhuang, T. Chang, R. Sivakumar, and A. Velayutham. A3: Application-Aware Acceleration for Wireless Data Networks. In *Proc. ACM MOBICOM*, 2006.