Real-time Bandwidth Prediction and Rate Adaptation for Video Calls over Cellular Networks

Eymen Kurdoglu
ek1666@nyu.edu

Yong Liu
yongliu@nyu.edu

Yao Wang
yw523@nyu.edu

Department of Electrical and Computer Engineering
NYU Tandon School of Engineering, Brooklyn, NY

Yongfang Shi
steveshi@tencent.com

ChenChen Gu
cicelygu@tencent.com

Jing Lyu
eckolv@tencent.com

Tencent Co. Ltd.
Shenzhen, China, 518057

ABSTRACT

We study interactive video calls between two users, where at least one of the users is connected over a cellular network. It is known that cellular links present highly-varying network bandwidth and packet delays. If the sending rate of the video call exceeds the available bandwidth, the video frames may be excessively delayed, destroying the interactivity of the video call. In this paper, we present Rebera, a cross-layer design of proactive congestion control, video encoding and rate adaptation, to maximize the video transmission rate while keeping the one-way frame delays sufficiently low. Rebera actively measures the available bandwidth in real-time by employing the video frames as packet trains. Using an online linear adaptive filter, Rebera makes a history-based prediction of the future capacity, and determines a bit budget for the video rate adaptation. Rebera uses the hierarchical-P video encoding structure to provide error resilience and to ease rate adaptation, while maintaining low encoding complexity and delay. Furthermore, Rebera decides in real time whether to send or discard an encoded frame, according to the budget, thereby preventing self-congestion and minimizing the packet delays. Our experiments with real cellular link traces demonstrate Rebera can, on average, deliver higher bandwidth utilization and shorter packet delays than Apple's FaceTime.

CCS Concepts

•Networks → Cross-layer protocols; Network experimentation; Network measurement; Mobile networks;

Keywords

real-time; cross-layer; forecasting; hierarchical-p

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MMSys'16, May 10-13, 2016, Klagenfurt, Austria

© 2016 ACM. ISBN 978-1-4503-4297-1/16/05…$15.00

DOI: http://dx.doi.org/10.1145/2910017.2910608

1. INTRODUCTION

Advances in networking and video encoding technologies in the last decade have made the real-time video delivery applications, including video calls and conferencing [17][9][3], an integral part of our lives. Despite their popularity in wired and Wi-Fi networks, real-time video applications have not found much use over cellular networks. The fundamental challenge of delivering real-time video over cellular networks is to simultaneously achieve high-rate and low-delay video transmission on highly volatile cellular links with rapidly-changing available bandwidth (ABW), packet delay and loss. On a cellular link, increasing the video sending rate beyond what is made available by the PHY and MAC layers leads to self-congestion, and intolerable packet delays, hence frame delays. Excessively delayed frames will have to be treated as lost. On the other hand, a conservative sending rate clearly leads to the under-utilization of the cellular channel and consequently a lower quality than what is possible.

The tight design space calls for a joint application and transport cross-layer design, involving real-time video encoding, bitrate control, sending rate adjustment, and error control. Ideally, the transmitted video rate should closely track the ABW on the cellular links. However, traditional *reactive congestion control algorithms* [30, 2], which adjust the sending rate through packet loss and/or packet delay information, are too slow to adapt to the changes in the ABW, leading to either bandwidth under-utilization or significant packet delays [27]. It is more preferable to design *proactive congestion control algorithms* that calculate the sending rate based on cellular link ABW forecasts. Meanwhile, for video adaptation, video encoder can adjust various video encoding parameters, so that the resulting video bitrate matches the target sending rate determined by the congestion control algorithm. However, accurate rate control is very challenging for low-delay encoding, and significant rate mismatch is often still present with the state-of-the-art video encoders. In addition, what makes the problem even more challenging is that the lost and late packets can render not only their corresponding frames, but also other frames non-decodable at the receiver. The encoder and the transport layer should be designed to be error resilient so that lost and late packets have minimal impact on the decoded video.

In this study, we propose a new real-time video delivery

system, Rebera, designed for cellular networks, where we aim to maximize the sending rate of the video source and error resilience, while keeping the one-way frame delays sufficiently small. Our system consists of a proactive congestion controller, a temporal layered encoder, and a dynamic frame selection module. Our proactive congestion controller uses the video frames themselves to actively measure the current ABW in real-time, and then employs the well-known linear adaptive filtering methods [11] to predict the future capacity, based on the past and present capacity measurements. For error resilience, we resort to layered encoding, which enables unequal error protection (UEP). However spatial and quality layering incurs significant encoding complexity and coding efficiency loss, making them unattractive for practical deployment. Thus we consider only temporal layering, which provides a certain level of error resilience even without using explicit UEP. To minimize the delays for real-time delivery, we use hierarchical-P (hierP) coding structure for temporal layering. To address the rate control inaccuracy of the encoder, we propose a dynamic frame selection algorithm for hierarchical-P, where the goal is to select in real-time which encoded frames to send, subject to a budget determined by the predicted capacity value. Our frame selection algorithm takes into account quality implications of the layers, decoding dependencies between the frames, and the smoothness of frame inter-arrivals to maximize the delivered video quality under the said budget. We have implemented the complete system, called "Rebera" for real-time bandwidth estimation and rate adaptation, to evaluate its performance and compare it with Apple's FaceTime video call application. Our implementation relies on an open source real-time H.264 video encoder [24], which we have modified to produce a hierarchical-P stream. As a result, we can directly control the encoded video rate according to the measured capacity. Thanks to the combination of all these components we have mentioned, Rebera is able to achieve higher bandwidth utilization and lower frame delays than FaceTime. In this study, we do not consider UEP among the temporal layers and the error resilience aspect of the system. These will be investigated in future studies.

2. RELATED WORK

Rate adaptation is a key problem for video transmission over best-effort networks. Most of the previous studies focus on one-way streaming of live or recorded video, where a few seconds of video buffering at the receiver can be tolerated. Due to buffering, a temporary mismatch between the video rate and the ABW does not directly impact the video playback, as long as the buffer does not drain out. The recent industry trend here is Dynamic Adaptive Streaming over HTTP (DASH) [20]. Various rate adaptation algorithms have recently been proposed [6, 22, 13, 29], and some of them were specifically designed for wireless networks, e.g. [19, 23].

To the contrary, video call involves two-way real-time video streaming. To facilitate live interaction, video call does not have the luxury of seconds of video buffering. Consequently, mismatch between the selected video rate and the ABW will directly translate into quality degradation of video playback, such as long video frame delays, delay jitters and video freezes. Thus, for a video call, real-time bandwidth estimation and video rate adaptation are more challenging, compared with one-way video streaming.

Sending rate determination, or congestion control, has been an active area of research for decades. Window-based congestion control algorithms are the dominant form of congestion control on the Internet, which reactively adjust the window size, and hence the sending rate according to a congestion signal. TCP variants such as Tahoe and New Reno [14] use packet losses, while Vegas [5], FAST [26] and Compound [21] react to packet round trip times. However, the additive-increase multiplicative-decrease (AIMD) probing method used in TCP, along with the retransmission of every single lost packet in these protocols renders them less desirable for interactive video calls. TFRC [8] and TCP Cubic [10] control the sending rate with smaller variations, however, their delay performances deteriorate in the face of fast ABW variations. As a result, rate-based congestion control protocols are dominantly used in commercial video call applications, such as Microsoft Skype, Google Hangouts and Apple FaceTime. Nonetheless, these protocols also behave reactively when adjusting the sending rate, and therefore suffer from the same self-congestion problem over highly volatile links. Authors of [27] proposed a proactive congestion control scheme for realtime video delivery in cellular networks. They model cellular links as single-server queues emptied out by a doubly-stochastic service process. For the ABW estimation, we, unlike [27], assume no particular time-evolution model for the link capacity. Furthermore, [27] focused only on congestion control without considering video adaptation.

Adapting the video rate in real-time according to the sending rate determined is crucial for a low-delay video application. This task is usually handled at the video encoder only. However, if the rate control is not accurate and the encoded video rate exceeds the rate constraint, sending every encoded frame will cause self-congestion. This problem can be alleviated if the video stream has temporal layering, by allowing the sender to prioritize from lower to higher layers, until the sending rate stays just below the rate constraint. However, on-the-fly decision of discarding a higher-layer frame that was encoded before the more important lower-layer frames is not trivial, since the sizes of the upcoming encoded frames are yet unknown. To the best of our knowledge, there is no published work addressing this problem.

3. PROPOSED SYSTEM OVERVIEW

We examine a real-time video delivery scenario between a sender and a receiver, where at least one user is connected to a cellular network (Figure 1). We denote the source device by S, the destination device by D, and the corresponding base stations by B_S and B_D, respectively. We call the directed path from S to D the forward path, and the directed path from D to S in the reverse direction the backward path. We assume that the in-network directed path (B_S, B_D) that connect the base stations has higher ABW, and constant queuing and propagation delay. Therefore, the overall ABW along the forward path (S, B_S, B_D, D) is equal to the minimum of the bandwidths along the cellular uplink (S, B_S) and cellular downlink (B_D, D).

According to the queuing model of [27], all packets destined to or sent from a given mobile device that is connected to a base station are queued up in isolated buffers, which are located on the mobile device for the uplink and in the base station for the downlink. These buffers are not shared by any other flow to or from other users; that is, there is no cross-traffic in these queues. The backlogged packets leave

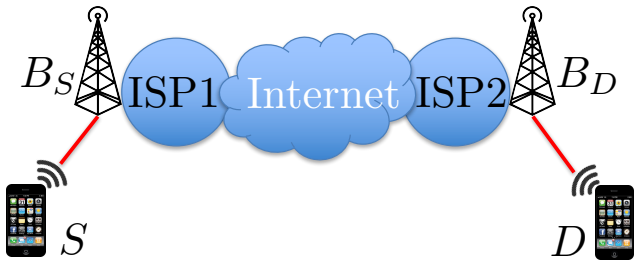

Figure 1: Cellular links between the mobile devices and their respective base stations are in red.

their respective buffers once they are successfully transmitted over the link. Thus, how fast these buffers are emptied out directly reflects the capacity of the cellular links, and consequently the end-to-end ABW.

As for the video stream, we assume that the sender uses a layered encoder so that it can easily adjust the sending rate by adjusting the number of video layers transmitted. Layered coding also enables unequal error protection; i.e., a basic level of quality can be guaranteed with high likelihood by providing more protection to the base layer. We consider only temporal layering to keep the encoding complexity and overhead at a minimum. In order to minimize the encoding delay, we further assume that the hierarchical-P structure (Figure 3) is used to achieve temporal scalability. Starting with the highest temporal layer, the frames can be discarded to reduce the video rate. In the example shown in Figure 3, each Group of Picture (GoP) consists of 4 frames, which leads to three temporal layers (TLs). We assume that the encoder inserts an I-frame every N frames, and we denote the time duration covering all N frames from an I-frame up to but excluding the next I-frame by an "intra-period." Then, the time duration T for an intra-period is equal to N/f, where f is the frame rate of the captured video.

We can now summarize the operation of the proposed system. Since rate control is usually performed once per intra-period in conventional video encoders, we predict the average ABW for each new intra-period. The prediction is based on the average ABWs for the past intra-periods, which are measured by the receiver and fed back to the sender. Specifically, the receiver periodically measures the ABW using the video frames that arrived within the last T-second window, and then feeds the result back to the sender. The window slides forward every Δ seconds. In order to have as fresh feedback messages as possible, we have $\Delta \ll T$. The sender, in turn, records the most recent measurement, and updates its value with the arrival of each new measurement. Then, at the beginning of the next intra-period k, the value of the most recent measurement is taken as the ABW $\tilde{c}_{k-1}$ measured during the last intra-period $k-1$. This value is input to an adaptive linear prediction filter, which then updates its prediction $\hat{c}_k$ regarding the ABW during the new intra-period k using the past bandwidth values $\tilde{c}_{k-1}, \ldots, \tilde{c}_{k-M}$. Using this prediction, the sender calculates the bit budget b_k, which is the maximum number of bits that the sender is allowed to send during this intra-period so that all the data that have been sent arrive at the receiver until the end of the intra-period with a high probability. The components of our design can be seen in Figure 2.

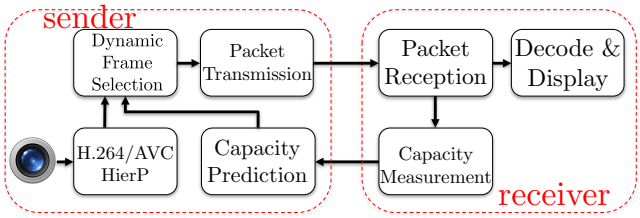

Figure 2: Proposed Rebera real-time video delivery system for cellular networks

4. SENDING RATE CONTROL

4.1 Measuring the Available Bandwidth

Packet pair/train methods [18] are well-known active capacity measurement schemes for finding the minimum capacity along a network path. The performance of these methods improve if there is no cross-traffic on the links, making them suitable to measure the cellular link capacity according to our model. In our system, we propose measuring the average ABW $c(t_1, t_2)$ actively at the destination, using the video frames received in $(t_1, t_2]$ as packet trains. Using the video frames as packet trains enables us to directly exploit the video data flow for capacity measurements and to avoid sending additional measurement traffic. Specifically, at the sender side, we first packetize each frame regardless of its size into $p \geq 2$ packets , and then send them together in a burst. The resulting instantaneous sending rate is likely to be higher than the instantaneous capacity of the link. As a result, packets queue up in the bottleneck; i.e. the base station buffer for the downlink or the cellular device buffer for the uplink, where they are transmitted one by one. Then, at the receiver side, we take capacity measurements $\{m_n\}$, where the sample m_n is obtained by using the arriving video frame n as a packet train. Let us denote the inter-arrival time between packet $i-1$ and i by a_i, and the size of the packet i by z_i. Then, we can calculate the capacity sampled by frame n as:

$$m_n \triangleq \frac{z_2 + \cdots + z_p}{a_2 + \cdots + a_p} \triangleq \frac{Z_n}{A_n}. \quad (1)$$

For any time period $(t_1, t_2]$, we can estimate the average capacity $c(t_1, t_2)$ over this time simply by

$$\tilde{c}(t_1, t_2) = \frac{\sum_{n \in \mathcal{N}} Z_n}{\sum_{n \in \mathcal{N}} A_n}, \quad (2)$$

where $\mathcal{N}$ is the set of all frames that arrived in $(t_1, t_2]$. Note that Eq. (2) is equivalent to taking a weighted average of all the capacity samples in $\{m_n\}$, where the sample m_n is weighted in proportion to its measurement duration A_n with weight $w_n = A_n / \sum_{n \in \mathcal{N}} A_n$. Having completed the average capacity measurement regarding $(t_1, t_2]$, the receiver prepares a small feedback packet and sends it back to the source. Note that we are ultimately interested in measuring the ABW c_k during $(T_k, T_{k+1}]$, where T_k denotes the start of the intra-period k. However, since the sender and receiver have different clock times in general, the receiver cannot know when exactly an intra-period starts. Furthermore, the feedback packets are subject to time-varying delays in the network. In short, we cannot guarantee that the feedback packets will arrive at the sender on time for predicting the

capacity of the next intra period. To address this issue, the receiver measures the average capacity within the last T seconds every Δ seconds, where $\Delta \ll T$. Each of these measurements are immediately sent back to the sender. Specifically, a measurement generated at time t is the average capacity in $(t-T, t]$, while the next measurement that is generated at $t+\Delta$ is the average bandwidth in $(t-T+\Delta, t+\Delta]$. The sender then uses the latest feedback received before T_k to predict the ABW in the next intra-period $(T_k, T_{k+1}]$. Lastly, assuming we keep the sending rate below the capacity, our measurement accuracy depends on the difference between the sending rate and the capacity of the link. If the sending rate equals, or by any chance, exceeds the capacity, we would have very high measurement accuracy, but this may lead to saturated links and long queueing delays, which are detrimental to video call quality.

Robustness against bursts

It is known that the cellular links occasionally experience channel outages that may last for up to several seconds, during which the capacity essentially drops to zero, and the packets in transit are backlogged in the respective buffers. As a result, the sender should stop sending any more packets as soon as an outage is detected. When the outage ends, all the packets queued up in the buffer are usually transmitted and arrive at the receiver as a burst. If the receiver uses these packets for capacity measurement, the burst rate, which is on the order of several Mbps, can severely disrupt the learning process of the predictor. In order to protect our system against these bursty measurements, we simply detect them through the sample measurement duration A_n. In our system, we consider a measurement bursty if $A_n < 10$ ms. Bursty measurements are simply discarded.

4.2 Predicting the Available Bandwidth

History-based forecast is a popular method for prediction [12], where the past measurement values are used to determine an estimate of the future. In this study, we perform linear prediction for history-based forecast. In particular, we chose a well-known online linear adaptive filter called the Recursive Least Squares (RLS) [11]. With each new capacity measurement regarding the last intra-period, RLS recursively updates its filter taps of length M, and makes a prediction for the capacity during the next intra-period. One of the advantages of the RLS algorithm is that it does not require a model for its input signal, and performs minimum least-squares regression [7]. Furthermore, it can adapt to the time-varying signal statistics through its forgetting factor λ, which serves to exponentially discount the weight of the past observations, without any need for a time-evolution model for the system. The notation regarding the RLS algorithm are summarized in Table 1.

The periodic prediction procedure is as follows. At $t = T_{k+1}$, which is the end of the intra-period k, the most recent capacity measurement received by the sender is taken as $\tilde{c}_k$, that is, the average ABW during the intra-period k. Then, the gain vector $\boldsymbol{g}(k)$ and the a priori prediction error ϵ_k are calculated, which are then used to update the filter tap vector $\boldsymbol{w}(k)$. At this point, the linear prediction for c_{k+1} is simply

$$\hat{c}_{k+1} = \boldsymbol{w}^T(k)\boldsymbol{c}(k). \tag{3}$$

The step concludes by updating the inverse of the empirical autocorrelation matrix of the measured capacities. The overall procedure is summarized in Algorithm 1.

Table 1: Notation regarding the RLS predictor

M	number of filter taps
λ	forgetting factor parameter
θ	initializer parameter for $\boldsymbol{P}$
$\boldsymbol{w}(k)$	filter tap vector of length M
$\boldsymbol{P}(k)$	inverse of empirical autocorrelation matrix, $M \times M$
$\boldsymbol{g}(k)$	gain vector of length M
$\tilde{c}_k$	measured capacity
$\boldsymbol{c}(k)$	vector of M most recently measured capacity values
ϵ_k	a priori prediction error

Algorithm 1 Recursive Least Squares

1: $\boldsymbol{P}(0) = \theta^{-1}\boldsymbol{I}, \boldsymbol{w}(0) = \boldsymbol{0}, \boldsymbol{c}(k) = \boldsymbol{0}$ ▷ *Initialization*
2: **for all** intra-period $k \geq 1$ **do**
3: $\quad \boldsymbol{g}(k) = \frac{\lambda^{-1}\boldsymbol{P}(k-1)\boldsymbol{c}(k-1)}{1+\lambda^{-1}\boldsymbol{c}^T(k-1)\boldsymbol{P}(k-1)\boldsymbol{c}(k-1)}$
4: $\quad \epsilon_k = \tilde{c}_k - \boldsymbol{w}^T(k-1)\boldsymbol{c}(k-1)$
5: $\quad \boldsymbol{w}(k) = \boldsymbol{w}(k-1) + \epsilon_k \boldsymbol{g}(k)$
6: $\quad \boldsymbol{P}(k) = \lambda^{-1}[\boldsymbol{P}(k-1) - \boldsymbol{g}(k)\boldsymbol{c}^T(k-1)\boldsymbol{P}(k-1)]$
7: $\quad \hat{c}_{k+1} = \boldsymbol{w}^T(k)\boldsymbol{c}(k)$
8: **end for**

4.3 Determining the Sending Rate

Our ultimate goal is to ensure that all the frames sent during an intra-period finish their transmission before the start of the next one. In other words, we aim to have each I-frame encounter empty buffers with high probability. Let us denote our sending rate in the intra-period $k+1$ by r_{k+1}. We determine r_{k+1} such that the probability to exceed the capacity c_{k+1} is low; that is,

$$\Pr(c_{k+1} < r_{k+1}) = \delta, \tag{4}$$

where δ is a small tolerance parameter that characterizes our tolerance to frequent ABW overshoots. Let ϵ_{k+1} denote the ratio of the actual capacity to the prediction obtained from the RLS algorithm, i.e, $\epsilon_{k+1} = c_{k+1}/\hat{c}_{k+1}$. Then, we can rewrite Eq. (4) as

$$\Pr\left(\epsilon_{k+1}\hat{c}_{k+1} < r_{k+1}\right) = \Pr\left(\epsilon_{k+1} < u_{k+1}\right) = \delta, \tag{5}$$

where $u_{k+1} \triangleq r_{k+1}/\hat{c}_{k+1}$ is referred to as safety coefficient. This means that, for a given δ value, r_{k+1} should be set by scaling the prediction $\hat{c}_{k+1}$ by u_{k+1}, the δ-quantile of ϵ_{k+1}. In Rebera, we set $\delta = 0.05$, and calculate the running 5-percentile of ϵ_{k+1} with a moving window [4].

Handling backlogged and lost packets

Note that, keeping the sending rate below the ABW cannot be guaranteed, even with the safety margin u_k, leading to occasional packet backlogs. If we do not consider the backlogged packets while determining the sending rate, the total number of bytes backlogged in the buffers accumulate in time. In order to address this issue, the sender, through information fed back by the receiver, estimates the number q_k of bytes still in the buffers at the end of the intra-period k, by subtracting the total number of bytes received at the receiver from the total number of bytes sent so far. However, in case of packet losses, q_k would keep growing in time, since

lost packets never arrive at the receiver. In order to account for the losses, we assume that the packets arrive at the destination in the order of their sequence numbers. To detect the number of lost bytes, we insert in each packet header the total number of bytes sent so far. Then, upon receiving a new packet, the receiver simply subtracts the number of bytes it has received so far from this number. The result is the number of bytes lost, which is fed back to the sender, along with the number of bytes received. The sender then determines q_k by taking the difference between the total number of bytes sent and the total number of bytes received and lost. Out-of-order packet deliveries will introduce only temporary errors to our estimates: after the delayed packets arrive at the receiver, our algorithm will automatically correct these errors in the next estimate. Combining all, we set the bit budget b_{k+1} for the intra-period $k+1$ as

$$b_{k+1} = (\hat{c}_{k+1} \times u_{k+1})T - q_k, \tag{6}$$

where T is the intra-period duration. *This way, we expect the network not only can finish the transmission of all video frames in intra-period $k+1$, but also can clean up the currently backlogged packets q_k by the end of intra-period $k+1$.*

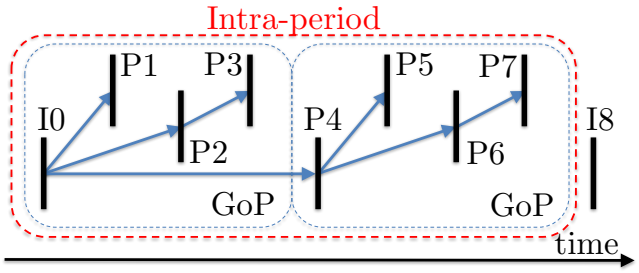

Figure 3: Hierarchical-P prediction. Blue arrows indicate the reference frames used to predict the frames being coded. In this example, N=8, G=4, TL0:[I0, P4]; TL1:[P2, P6]; TL2:[P1, P3, P5, P7]

5. REAL-TIME FRAME SELECTION FOR HIERARCHICAL-P VIDEO

Video rate control is crucial for real-time applications over networks with time-varying bandwidth. However, accurate rate control is very challenging, especially in the very low-delay scenarios, where look-ahead and multi-pass encoding are not suitable. In spite of the extensive research in this area [16], significant mismatch between the target and actual bitrate over an intra-period can still occur [16]. For example, when using the well-designed x264 encoder [24], we have observed up to 25% rate mismatch in the presence of sudden motion when the video is coded using the IPPP, as well as the hierarchical-P structure. In case of rate mismatch, if the video is coded with the IPPP structure, all remaining frames will have to be discarded once the target budget for an intra-period is used up. When this happens early in the intra-period, the receiver experiences a relatively long freeze.

To remedy this problem, we propose to use a temporal layered encoder with the hierarchical-P coding structure, so that the sending rate can be adjusted by skipping the higher layer frames, without incurring additional encoding delay or complexity. Figure 3 shows an example prediction structure for the hierarchical-P encoding, which yields three temporal layers. We propose a frame selection scheme that either discards or sends each encoded frame, subject to the given bit budget b_k and frame dependencies. We assume that the video encoder runs its own rate control algorithm, but may not meet the bit budget per intra-period accurately. When the encoded bitrate exceeds the budget, an encoded frame may be dropped by the frame selection scheme so that the actual sending rate never exceeds the predicted bandwidth for an intra-period. The benefit of using the hierarchical-P structure is that the delivered video has more evenly-spread frames, whereas the IPPP structure can lead to a very jittery video when some frames are dropped. With the frame selection module outside the encoder, the encoder rate control can be less conservative. This, in turn, can lead to higher bandwidth utilization.

5.1 Dynamic Frame Selection

Frame selection is ultimately about allocating the budget for more important (lower layer) frames. Higher layer frames can be sent only if there is available bit budget after sending the lower layer frames. However, to minimize the delays, the decision to either send or discard a given frame must be made right after it is encoded, without knowing future frame sizes. For example, in Figure 3, we cannot wait to see if we can send P4 first, followed by P2 and then P1. Rather, we have to determine whether we send P1 as soon as P1 arrives. If the future frames from lower layers are large, sending frames from a current higher layer may preclude the sending of upcoming lower layer frames. On the other hand, dropping frames from higher layers when the future lower layer frames are small would underutilize the ABW.

Given an intra-period, let us label each frame with its appearance order, and denote the size and the temporal layer of the frame n by s_n and ℓ_n, respectively. Our goal is to decide, for each encoded frame n, to either send or discard it, such that the total number of frames sent at the end of the intra-period is maximized, while the mean and the variance of the time gap between the selected frames are kept small. We start our frame selection procedure by estimating frame size for each temporal layer, in order to make decisions considering the future frames. We then continue by ordering the frames in this intra-period based on their layer numbers, starting with the lowest layer, since the higher layer frames cannot be decoded with the lower layers. We denote this priority order by an ordered list π. For each newly arriving frame n, we trim π into π_n by excluding the past frames for which a decision has already been made, and the future frames that cannot be decoded at the receiver due to the previously discarded frames. π_n is basically the priority order among the eligible frames left. Next, we update the frame size estimations, as well as our estimation for the remaining bit budget. Then, we create a set E_n of frames that we expect to send according to our frame size and the remaining bit budget estimations, by greedily picking frames starting from the first frame in π_n. We stop picking the frames when the total estimated size of the frames picked reaches the estimated bit budget. Finally, if frame n is in the set E_n, we send it; otherwise it is discarded.

For frame size estimation, we assume that the frame sizes in the same temporal layer will be similar. Therefore, we keep a frame size estimate $\hat{s}_\ell$ for each layer ℓ. In this study, we use an exponentially weighted moving average (EWMA)

filter with parameter γ for estimating the size of future frames in layer l using the actual sizes of the past coded frames in this layer. Note that for the base layer, we apply the above method only to the successive P-frames as the I-frame size is much larger than P-frames. We do not need to estimate the I-frame size, since we always send the I-frames. The overall algorithm is summarized in Algorithm 2.

5.2 Bit Budget Update

The bit budget b_k is the estimation of the total number of bits that the sender can transmit during the intra-period k without causing buffer build-up. Here, we assume that, at any time t since the start of the intra-period, $\frac{t}{T}b_k$ bits can be transmitted on average, with a mean rate of b_k/T. Thus, if the sender sends less than this amount, the unused bandwidth is wasted. In order to account for these missed transmission opportunities, we update the remaining bit budget at each step n by

$$\hat{b}_k(n) = b_k - \max\left(S_n, \frac{n}{N}b_k\right), \tag{7}$$

where S_n is the total number of bits sent before selecting frame n. Without updating the budget, the sender may end up sending large frames close to the end of the intra-period, which would then backlog in the buffer, and potentially delay all the packets in the next intra-period.

5.3 Frame Priority Order

In the frame priority list π, placing frame i before frame j means we allocate our bit budget to send frame i first, and that frame j is sent only if there is sufficient bandwidth budget to do so, after we have decided to send all the frames placed before frame j. Accordingly, lower layer frames have higher priority than the higher layer frames, which depend on the former. Within the base layer, the frames are ranked in their encoding order, as they follow the IPPP coding structure. However, within an enhancement layer, *any* order of frames is decodable, since the frames from lower layers are picked before. Now, if the layer l frames are prioritized sequentially from the beginning, budget depletion results in a lower frame rate until the intra-period ends. On the other hand, if the frames are prioritized starting from the end, we may miss the transmission opportunities for the earlier frames, if the latter frames turn out smaller. Therefore, we pick the frames in multiple steps, alternating the direction in each step to strike a balance. Starting with the list of frames in the appearance order, we divide the list into two equal-sized lists at each step. We then pick the last frame from each smaller list, following the direction at that step.

6. SIMULATIONS AND EXPERIMENTS

6.1 Forecasting via Adaptive Filtering

We start our evaluations by motivating the use of the RLS linear adaptive filter for capacity prediction. We compare the prediction performance of the RLS with the simple and popular EWMA predictor [12]. In our experience, the filter length and the forgetting factor parameters do not significantly affect the prediction errors provided that we choose $M < 10$ and $\lambda > 0.99$. Therefore, we have selected $M = 5$, $\lambda = 0.999$, $\theta = 0.001$ and fixed this configuration for the rest of the evaluations. We collected six real cellular link capacity traces (Figure 7) following the methodology in [27], over

Algorithm 2 Dynamic Frame Selection

```
S_0 = 0, π_0 = π, intra-period k, bit budget b_k
for all frames n = {0, ..., N − 1} do
    ŝ_j ← γ s_n + (1 − γ) ŝ_j, for each frame j ∈ ℓ_n
    b̂_k(n) = b_k − max(S_n, (n/N) b_k)
    Create E_n from π_n, based on ŝ and b̂_k(n)
    if n ∈ E_n then
        S_{n+1} = S_n + s_n and send frame n
    else
        π_{n+1} = π_n − {frames depending on n}
    end if
    π_{n+1} = π_n − n
end for
```

Table 2: Statistics of traces used in the experiments.

	Mean (kbps)	Std (kbps)	Coeff. of Var.	Outage %
Tr1	176	115	0.654	2.0
Tr2	388	165	0.425	0.5
Tr3	634	262	0.413	0.0
Tr4	735	264	0.359	0.2
Tr5	937	356	0.379	1.2
Tr6	1055	501	0.475	0.1

T-Mobile 3G and HSPA networks, during different times of the day and in different campus locations. Each of these is 1066 seconds long and their statistics can be found in Table 2. As expected, the capacity traces are very dynamic, posing significant challenge to capacity estimation.

Over these traces, in Matlab, we perform time-series forecasting using RLS with parameters mentioned above, and the EWMA filter, where the smoothing parameter α is varied from 0 to 1. We assume that we know the past capacity values exactly. The results can be seen in Table 3, where "Best" and "Worst" represent the minimum and maximum prediction error root-mean square (RMS) values obtained with EWMA with different smoothing parameters, respectively. We see that for all traces, prediction performance of RLS is very close to that of the best EWMA predictor, if not better, as it adapts to the statistics of the time series.

6.2 Dynamic Frame Selection Simulations

Next, we compare the performance of our dynamic frame

Table 3: Comparing the prediction error RMS of the RLS predictor with those of the best and worst EWMA predictors with corresponding parameters. RLS, Best and Worst columns are in Kbps.

	RLS	α_{Best}	Best	α_{Worst}	Worst
Tr1	53	0.55	55	0.05	87
Tr2	88	0.7	90	0.05	120
Tr3	158	0.55	157	0.05	209
Tr4	186	0.4	178	0.05	211
Tr5	250	0.2	235	1	293
Tr6	244	0.4	242	0.05	291

selection (DFS) algorithm against the layer-push (LP) and frame-push (FP) algorithms. LP also estimates the frame size in each temporal layer using the same approach as in DFS, but then decides on the highest layer $l_{\max}$ that may be sent. In other words, only the frames from layers up to $l_{\max}$ are eligible for sending. Among these frames, following the encoding order, the algorithm sends as many frames as possible until the bit budget is exhausted. FP, on the other hand, does not consider layer information and sends as many frames as possible following their encoding order, until the bit budget is exhausted.

For each algorithm, we evaluate the total number of frames sent, the mean and the standard deviation of the resulting frame intervals, and finally the fraction of the unused bit budget. Here, a frame interval represents the temporal distance between a pair of consecutive frames that have been selected to be sent. The frame interval statistics are calculated using the fraction of time each interval lasts as the probability to observe that interval. We use the JM encoder [15] to encode the video sequence "Crew" [28] with a hierarchical-P structure having three temporal layers (GoP length=4) and intra-period of 32 frames. We used a fixed quantization parameter (QP) of 36, yielding the average bitrate of 415 kbps when all frames are included. The resulting video sequence has a frame rate of 30 fps and comprises 9 intra-periods, with an intra-period of $T = 32/30$ seconds. For the proposed algorithm, we used $\gamma = 0.75$, which was found to perform the best, and the frame priority order is $\pi = (0, 4, 8, 12, 16, 20, 24, 28, 30, 14, 6, 22, 26, 18, 10, 2, 31, 15, 7, 23, 27, 19, 11, 3, 1, 5, 9, 13, 17, 21, 25, 29)$. In these simulations, we assume that the bit budget b_k is constant for each intra-period k of the video and we want to compare the performances of the algorithms described above under different b_k values, from 10 kB to 80 kB. In Figure 4, we see that FP sends the most frames by sending as many frames as possible. However, it also has the largest mean frame interval and the largest frame interval variation, making the displayed video jittery. On the other hand, the LP algorithm sends the lowest number of frames but also with lower mean frame interval and frame interval variance. The proposed DFS algorithm achieves a good compromise between sending more frames, consequently utilizing ABW more closely, and reducing the frame distance variation. In fact, DFS outperforms both methods in terms of the mean and standard deviation of the frame intervals, while sending almost as many frames as the FP. Finally, the plot in the upper right shows the fraction of the unused bandwidth for each method, where we see that the performance of DFS is very similar to FP, whereas LP is not as efficient.

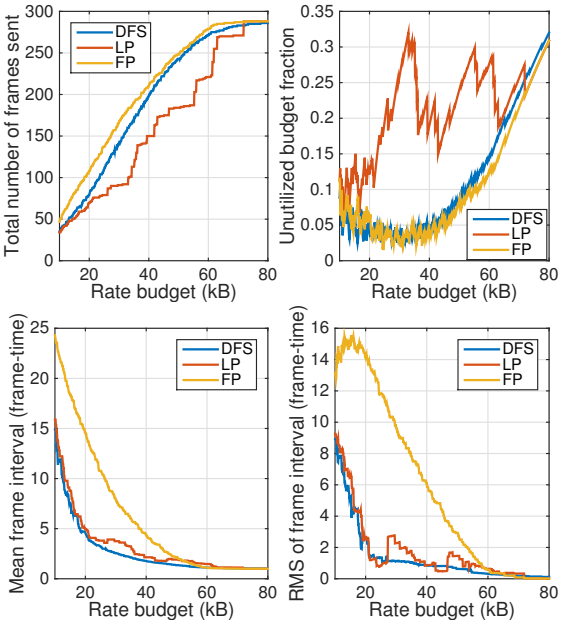

Figure 4: Comparison of DFS with FP and LP; number of frames sent (upper-left), unused budget (upper-right), mean and standard deviation of the frame intervals (lower-left and lower-right). Encoding frame rate is 30 Hz, thus frame-time is 1/30 sec.

6.3 Evaluation on the Testbed

For system evaluation, we developed a testbed to compare Rebera with popular video call applications. On this testbed (Figure 5), S and D are the source and destination end-points running the video call application under test, while the nodes C_S and C_D are cellular link emulators running the CellSim software [27], respectively. The emulators are connected to each other through the campus network, and to their respective end-points via Ethernet. For cellular link emulation, we use the uplink and downlink capacity traces collected (Table 2). For evaluation, we report the ABW utilization, the 95-percentile one-way packet delays, and the 95-percentile one-way frame delays as the performance metrics. In order to calculate the bandwidth utilization, we count how many bytes were sent out by the video call application under test and compare it with the minimum of the capacities of the sender link and the receiver link. The one-way end-to-end delays are collected by different means: in Rebera experiments, for each packet that made it to the receiver, the receiver sends an acknowledgement packet back to the sender over an ethernet cable on which there is no other traffic (Figure 5). As a result, the measured round-trip times are almost equal to the one-way delays, enabling us to measure the delay for each individual packet and frame. In FaceTime experiments, we used Wireshark to sniff the packets on the emulator hosts. We also note that FaceTime sends voice packets even after the voice is muted, at a constant rate of 32 kbps. Rebera, on the other hand, does not send audio. In order to compensate for this in the bandwidth utilization calculations, we subtract 32 kbps from the sending rate achieved by Rebera. In each test, we loop the video sequence "Crew", which is more challenging in terms of the video rate than "Akiyo" and somewhat captures hand/arm movements present in video calls.

Rebera is able to encode the video in real-time thanks to the open source x264 video encoder [24]. This allows us to change the video rate according to the predicted ABW, for each new intra-period, using x264's rate control module. We have modified x264's code, so that the encoded video has a hierarchical-P coding structure, by changing the reference frames used before encoding each frame according to the H.264/AVC standard. Specifically, in our modification, the GoP length is set to 4, giving rise to 3 temporal layers. In all our experiments in the lab, the minimum and maximum encoding rates were set as 200 kbps and 3 Mbps, respec-

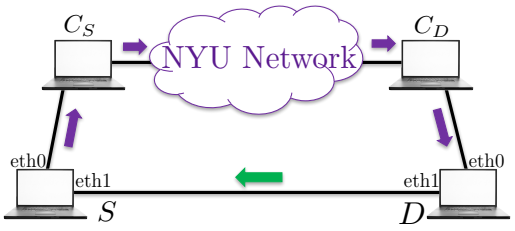

Figure 5: Illustration of the testbed. Purple arrows indicate the flow direction of the video, whereas the acks follow the green arrow from D to S.

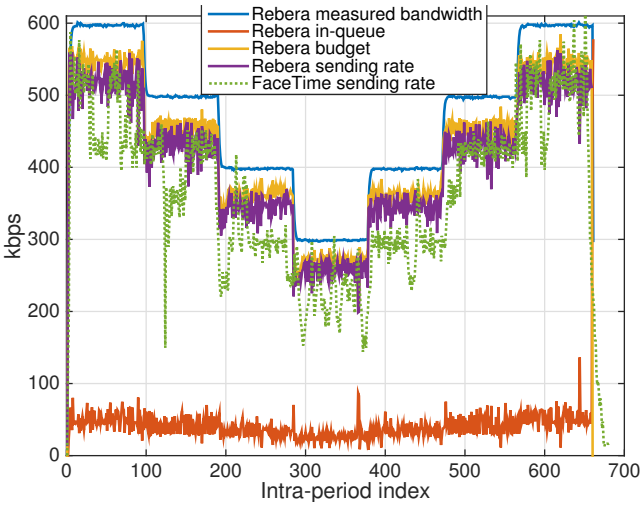

Figure 6: Sending rate of Rebera and FaceTime under piecewise constant bandwidth.

tively. The video and RLS parameters used are the same as in Sections 6.2 and 6.1. Specifically, the encoding frame rate is 30 Hz, and the intra-period length T is 32 frames, or 1.066 seconds. The initial sending rate is set to 120 kbps. In each experiment, we evaluate the sending rate over consecutive periods of T seconds. Please note that FaceTime may not be using a constant intra-period, let alone the same intra-period T as Rebera. Furthermore, FaceTime's sending rate is, in general, the sum of FEC and the video data rates. In order to feed the same looped test video into FaceTime, we used the ManyCam [25] virtual webcam on Mac OS 10.10.4[1].

6.3.1 *Evaluation with Piecewise Constant Bandwidth*

In this experiment, we use a piecewise constant bandwidth trace, with steps of 100 kbps lasting 100 seconds, between 300 and 600 kbps. We set the packet loss rate to zero. In Figure 6, we can see Rebera's (i) measured bandwidth, (ii) rate reduction due to the estimated number of backlogged bits, (iii) overall budget and (iv) the sending rate, along with FaceTime's sending rate. On average, the bandwidth utilization of Rebera is 86.21%, while FaceTime achieves a utilization of 78.78%. Moreover, we can observe that, Rebera is able to measure the current bandwidth very accurately, and thus react to the changes in the bandwidth rapidly.

6.3.2 *Evaluation with Cellular Capacity Traces*

[1]Detailed explanations can be found online at [1].

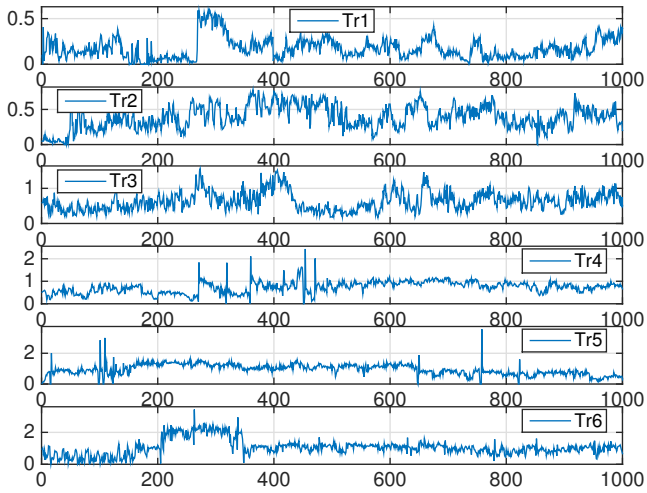

Figure 7: Traces used in the experiments. Vertical axis: capacity (Mbps), horizontal axis: intra-period index. Traces 2, 3, 4 and 5 are used as forward capacities, 1 and 6 are used as backward capacities.

In this set of experiments, we use cellular bandwidth traces (Figure 7) to emulate the cellular links. Each experiment lasts for 1000 intra-periods (1066 sec). We first present the results involving a single cellular link along the end-to-end path. Specifically, we start by examining the particular scenario where the sender is connected through a cellular network, and traces 5 and 6 were used to emulate the forward and backward end-to-end ABW, respectively. The receiver is assumed to have a wired connection. In the top plot of Figure 9, we present Rebera's and FaceTime's sending rates over time. Here, Rebera achieves a forward bandwidth utilization of 75.6%, while FaceTime's utilization is 65.2%. Furthermore, 95-percentile packet and frame delays are observed to be 204 and 232 ms for Rebera, and 307 and 380 ms for FaceTime. The empirical packet and frame delay CDFs for both systems can be seen in Figure 8.

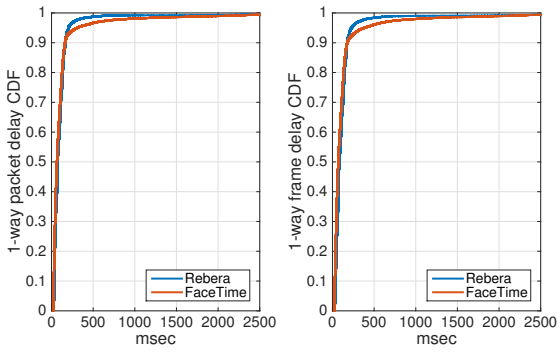

Figure 8: Empirical packet (left) and frame (right) delay CDFs of Rebera and FaceTime, where forward and backward bandwidths were emulated via traces 5 and 6, respectively.

Similarly, we employ traces 2, 3, 4 and 5 as the forward, and traces 1 and 6 for the backward end-to-end ABW. Results are summarized as bandwidth utilization, 95-percentile

Table 4: Evaluation over single cellular link, using Trace 1 as the backward capacity. Reported values are bandwidth utilization percentage, 95-perc. packet delay, and 95-perc. frame delay, respectively.

Fwd Cap.	**Rebera(%,ms,ms)**	**FaceTime(%,ms,ms)**
Trace 2	68.8, 402, 402	58.1, 447, 415
Trace 3	63.8, 338, 334	32.7, 631, 528
Trace 4	73.5, 177, 201	63.0, 383, 392
Trace 5	71.8, 216, 243	58.7, 317, 341
Average	69.5, 283, 295	53.1, 444, 419

Table 5: Evaluation over single cellular link, using Trace 6 as the backward capacity. Reported values are bandwidth utilization percentage, 95-perc. packet delay, and 95-perc. frame delay, respectively.

Fwd. Cap.	**Rebera(%,ms,ms)**	**FaceTime(%,ms,ms)**
Trace 2	69.5, 381, 394	59.2, 426, 406
Trace 3	66.4, 307, 313	61.9, 341, 349
Trace 4	76.1, 168, 189	70.6, 276, 307
Trace 5	75.6, 204, 232	65.2, 307, 380
Average	71.9, 265, 282	64.2, 337, 360

packet and frame delay tuples in Tables 4 and 5. We can see from Table 4 and 5 that in all experiments, Rebera achieves a higher utilization of the forward ABW with shorter delays. Averaged over these experiments, Rebera provides 20.5% higher bandwidth utilization compared to FaceTime, and a reduction of 122 ms and 102 ms in the 95-percentile packet and frame queueing delays, respectively. When a more challenging backward capacity (trace 1 in Table 4) is used for the backward path, the information fed back to the sender side undergo a longer delay for both Rebera and FaceTime, decreasing the ABW utilization of both systems. FaceTime's delay performance also degrades, whereas Rebera is still able to provide similar delays.

Next, we consider the scenarios where both users are connected over different cellular links. We assume there exists three different cellular connections, which we denote by A, B and C, where the uplink and downlink ABW pairs for each connection are given as traces 5 and 6, traces 3 and 4, and traces 1 and 2, respectively. In other words, connection A provides the highest mean ABW, while the connection C provides the lowest. We evaluate all six cases for which the sender and the receiver have different connections. The results can be seen in Table 6. In all scenarios, Rebera provides a significantly higher ABW utilization, while still delivering shorter packet and frame delays on average and in most cases.

6.3.3 *Effect of the Tolerance Parameter*

Next, we investigate the effect of the tolerance parameter δ in Section 4.3 on Rebera. We vary δ from 0.05 up to 0.5, and record the utilization and 95-percentile packet delays in Table 7. Having a larger δ value means the system is willing to tolerate more frequent capacity overshoots, and hence more frequent large packet and frame delays, in exchange for higher bandwidth utilization, which could be the case for video applications with less stringent delay requirements.

Table 6: Evaluation when both users are connected over different cellular networks. Reported values are bandwidth utilization percentage, 95-perc. packet delay, and 95-perc. frame delay, respectively.

	Rebera(%,ms,ms)	**FaceTime(%,ms,ms)**
A to B	60.5, 300, 312	47.9, 529, 508
B to A	58.1, 483, 498	48.5, 485, 483
A to C	59.9, 432, 442	44.0, 588, 518
C to A	61.3, 1066, 1019	43.3, 1278, 851
B to C	61.2, 416, 419	28.2, 1180, 996
C to B	59.5, 804, 809	44.0, 3230, 1090
Average	60.1, 583, 583	42.6, 1215, 741

Table 7: Effect of the tolerance parameter on Rebera over single cellular link. Forward-backward capacity: traces 3-6

δ	0.05	0.1	0.2	0.5
ABW utilization (%)	66.4	69.6	72.7	75.36
95-perc. packet delay (ms)	307	354	371	486
95-perc. frame delay (ms)	313	367	403	516

6.3.4 *Rebera Bahavior in Presence of Packet Loss*

The purpose of this evaluation is to demonstrate that Rebera can still track the link capacity in the presence of packet loss. Note that additional studies are needed to investigate the error resilience provided by the temporal layering, and how to further improve it through unequal error protection. To examine the performance of Rebera in presence of packet loss, we employ CellSim to introduce random iid losses. We tested Rebera when the packet loss rate is 5% and 10%, and the results are given in Table 8. Although not significantly, the ABW utilization drops with the loss rate, as there are fewer packets crossing the links. Furthermore, the delays experienced by the received frames reduce, since there is less backlog in the buffers.

Table 8: Effect of the packet losses on Rebera over single cellular link. Forward-backward capacities: traces 3-6

Packet loss rate	0%	5%	10%
ABW utilization (%)	66.4	63.4	61.1
95-perc. packet delay (ms)	307	281	286

6.4 Evaluation over Cellular Networks

Finally, we evaluate Rebera over a real cellular network. The setup we used for this experiment can be seen in Figure 10. Here, a mobile device (Motorola Nexus 6) is tethered to the sender host via USB, acting as a modem. The sender

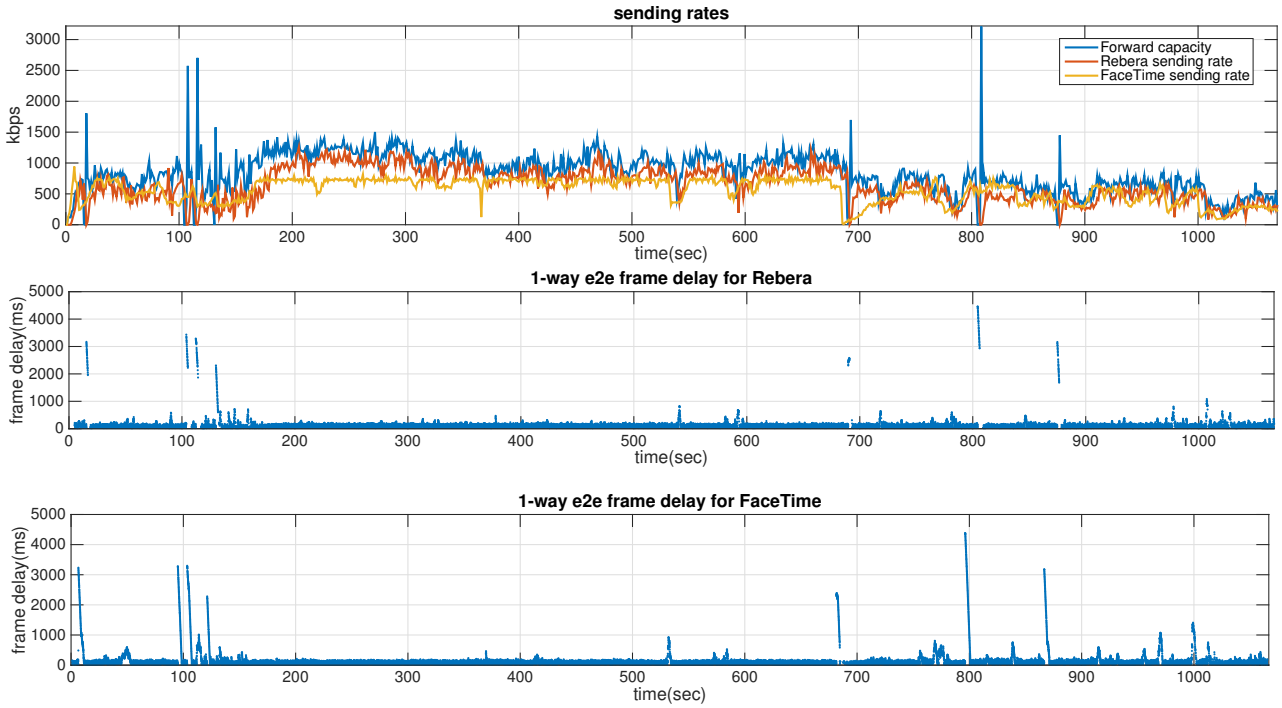

Figure 9: Video sending rates of Rebera and FaceTime (top) over a single cellular link with traces 5 & 6 as the forward & backward capacities, respectively. We also present the one-way end-to-end frame delays of Rebera (middle) and FaceTime (bottom), where the horizontal axis represents the sending time of each frame. The average ABW utilization and 95-perc. packet and frame delay are reported in Table 5, row 4.

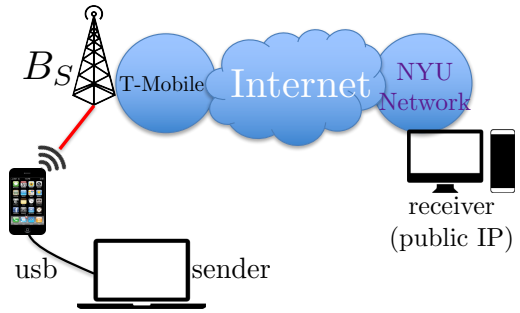

Figure 10: Setup for experiments in-the-wild

is stationary during the experiments, which last for 10 minutes. The receiver host is inside the campus network, and has a public IP address. The experiment was done over the T-Mobile network, using LTE and HSPA on December 4, 2015 at 6 PM. In Figure 11, we can see that, in the HSPA uplink, which provides an average ABW of 1.055 Mbps, the outages may last as long as 20 seconds. On the other hand, LTE provides an almost outage-free uplink channel, with an average ABW of 5.95 Mbps. Table 9 summarizes the experiment results. Note that during the experiment over LTE, Rebera's maximum encoding rate is set as 10 Mbps. This change serves as a means to utilize the ABW better, and is not necessary for Rebera's operation. The empirical packet delay distributions for Rebera using either access technology is given in Figure 12.

Table 9: Rebera's performance over T-Mobile network with HSPA and LTE technologies

Rebera	HSPA	LTE
average sending rate (Mbps)	0.81	5.17
95-perc. packet delay (ms)	221	105
95-perc. frame delay (ms)	298	137

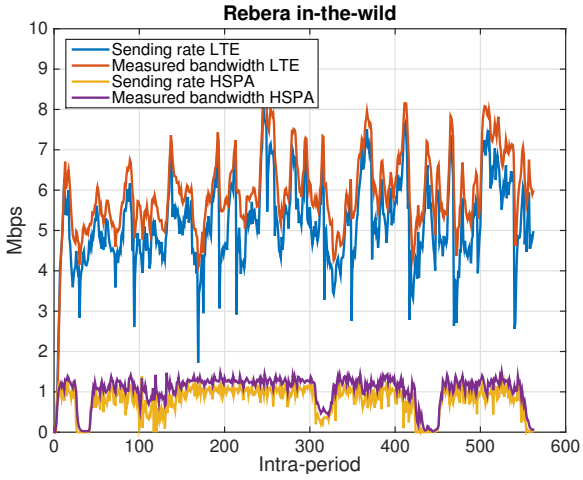

Figure 11: Rebera sending rate and measured bandwidth over LTE and HSPA

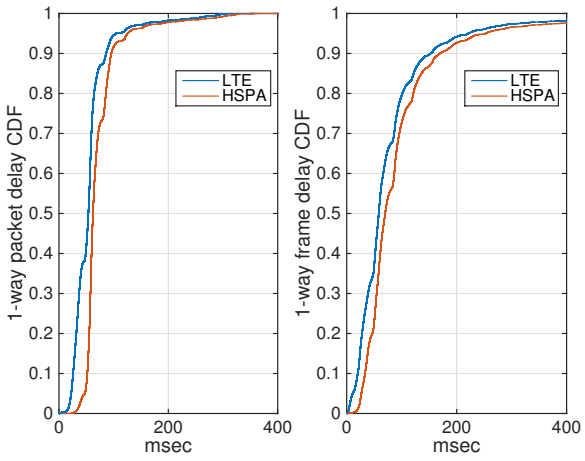

Figure 12: One-way packet and frame delay CDFs of Rebera over LTE and HSPA

7. CONCLUSIONS

Video calls over cellular links have to adapt to fast-changing network bandwidth and packet delay. In this study, we proposed a new real-time video delivery system, Rebera, designed for cellular networks. Rebera's proactive congestion controller uses the video frames to actively measure the capacity of cellular links, and through these measurements makes a safe forecast for future capacity values, using the well-known adaptive filtering techniques. Through its dynamic frame selection module designed for temporal layered streams, Rebera ensures that its video sending rate never violates the forecast by discarding higher layer frames, thereby preventing self-congestion, and reducing the packet and consequently the frame delays. Our experiments showed that Rebera is able to deliver higher bandwidth utilization and shorter packet and frame delays compared with Apple's FaceTime on average. In the future, we will consider UEP among temporal layers to improve the system performance in the presence of packet loss.

8. ACKNOWLEDGMENTS

This research is supported in part by Tencent Co. Ltd, the NYU WIRELESS center, and the New York State Center for Advanced Technology in Telecommunications (CATT).

9. REFERENCES

[1] Video Lab @NYU. http://vision.poly.edu/index.html/index.php?n=HomePage.Rebera.

[2] H. Alvestrand and S. Holmer. A Google Congestion Control Algorithm for Real-Time Communication on the World Wide Web. 2012.

[3] Apple. Facetime. http://www.apple.com/mac/facetime/.

[4] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *Proceedings of ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 286–296. ACM, 2004.

[5] L. S. Brakmo and L. L. Peterson. TCP Vegas: End-to-end Congestion Avoidance on a Global Internet. *IEEE JSAC*, 13(8):1465–1480, 1995.

[6] L. De Cicco, S. Mascolo, and V. Palmisano. Feedback control for adaptive live video streaming. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 145–156. ACM, 2011.

[7] B. Farhang-Boroujeny. *Adaptive filters: theory and applications*. John Wiley & Sons, 2013.

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

[9] Google. Hangouts. http://www.google.com/+/learnmore/hangouts/.

[10] S. Ha, I. Rhee, and L. Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[11] S. Haykin. Adaptive filter theory. *Prentice Hall*, 2002.

[12] Q. He, C. Dovrolis, and M. Ammar. On the predictability of large transfer tcp throughput. *ACM SIGCOMM Computer Communication Review*, 35(4):145–156, 2005.

[13] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 187–198. ACM, 2014.

[14] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM computer communication review*, volume 18, pages 314–329. ACM, 1988.

[15] JM Ref. Software. http://iphome.hhi.de/suehring/tml/.

[16] Y. Liu, Z. G. Li, and Y. C. Soh. A Novel Rate Control Scheme for Low Delay Video Communication of H.264/AVC Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(1):68–78, 2007.

[17] Microsoft. Skype. http://www.skype.com.

[18] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *IEEE Network*, 17(6):27–35, 2003.

[19] W. Seo and R. Zimmermann. Efficient video uploading from mobile devices in support of http streaming. *ACM Multimedia Systems*, 2012.

[20] I. Sodagar. The MPEG-DASH Standard for Multimedia Streaming over the Internet. *IEEE MultiMedia*, (4):62–67, 2011.

[21] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP Approach for High-speed and Long Distance Networks. In *Proceedings of IEEE INFOCOM*, 2006.

[22] G. Tian and Y. Liu. Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming. In *ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2012.

[23] G. Tian and Y. Liu. On Adaptive Http Streaming to Mobile Devices. In *Packet Video Workshop*. IEEE, 2013.

[24] VideoLAN. x264. http://www.videolan.org/developers/x264.html.

[25] Visicom Media. ManyCam. https://manycam.com.

[26] D. X. Wei, C. Jin, S. H. Low, and S. Hegde. FAST TCP: motivation, architecture, algorithms, performance. *IEEE/ACM Transactions on Networking (ToN)*, 14(6):1246–1259, 2006.

[27] K. Winstein, A. Sivaraman, and H. Balakrishnan. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 459–471, 2013.

[28] Xiph.org. Test Media. https://media.xiph.org/video/derf/.

[29] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication*, pages 325–338. ACM, 2015.

[30] X. Zhang, Y. Xu, H. Hu, Y. Liu, Z. Guo, and Y. Wang. Profiling Skype Video Calls: Rate Control and Video Quality. In *Proceedings of IEEE INFOCOM*, 2012.