Congestion-aware MAC Layer Adaptation to Improve Video Teleconferencing over Wi-Fi

Wei Chen
InterDigital
San Diego, CA 92121, USA
wei.chen@interdigital.com

Liangping Ma
InterDigital
San Diego, CA 92121, USA
liangping.ma@interdigital.com

Chien-Chung Shen
Computer and Information Sciences
Univ. of Delaware, DE 19716, USA
cshen@udel.edu

ABSTRACT

In wireless networks such as those based on IEEE 802.11, packet losses due to fading and interference are often misinterpreted as indications of congestion, causing unnecessary decrease in the data sending rate due to congestion control at higher layer protocols. For delay-constrained applications such as video teleconferencing, packet losses may result in excessive artifacts or freeze in the decoded video. We propose a simple and yet effective mechanism to detect and reduce channel-caused packet losses by adjusting the retry limit parameter of the IEEE 802.11 protocol. Since retry limit is left configurable in the IEEE 802.11 standard, and does not require cross-layer coordination, our scheme can be easily implemented and incrementally deployed. Experimental results of applying the proposed scheme to a WebRTC-based realtime video communication prototype show significant performance gain compared to the case where retry limit is configured statically.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication; C.2.3 [**Network Operations**]: Network management, Public networks; H.5.1 [**Multimedia Information Systems**]: Video.

General Terms

Design, Algorithms, Performance

Keywords

Wi-Fi, 802.11 MAC, retry limit, WebRTC, video teleconferencing, congestion detection, Google Congestion Control(GCC)

1. INTRODUCTION

Wi-Fi (IEEE 802.11) networks have been widely deployed and the adoption is still fast growing. However, in real-time video applications, such as video teleconferencing over time-varying, error-prone and bandwidth-fluctuating channels, Wi-Fi networks still face many challenges. A typical real-time video system may have Wi-Fi link(s) (shared with multiple competing peers) on the edge and Internet in the core as shown in Fig. 1, where the video sender transmits encoded video data to the video receiver for decoding and playback. Standard video codecs (such as H.264 and VP8) exploit the spatial and temporal redundancy in uncompressed video to achieve a high compression ratio, which, however, makes compressed video very sensitive to transmission errors. Packet losses due to transmission errors often lead to serious video quality degradation, like artifacts in the decoded video, which affects not only the current frame, but also subsequent frames because of error propagation resulted from the use of prediction from previous frames. Some error concealment techniques can help stop error propagation, for example frame copy that is used in WebRTC [20] where a frame that cannot be correctly decoded is replaced by the last correctly decoded frame. This however causes video freezes. Application layer Forward Error Correction (FEC) is a commonly used scheme for error protection. However, it introduces extra complexity, overhead and delay, which is undesirable to applications with stringent delay constraints like video teleconferencing. In section 2, we will investigate the efficiency problem when FEC is used in our WebRTC-based testbench. One straightforward and preferable solution is to develop a lightweight mechanism to reduce packet losses caused by channel errors, which is the focus of this paper.

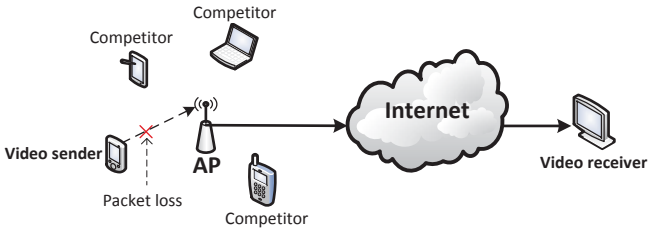


Figure 1: A typical real-time video system

In wireless networks, path loss, shadowing, fading and interference cause packet losses. Packet losses due to these reasons are classified as channel-caused losses. Note that such packet losses, which occur when the distance between a wireless station and an Access Point (AP) increases or when obstacles move temporarily between the station and the AP, are very frequent in Wi-Fi networks. When the channel

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 '15, March 18 - 20, 2015, Portland, OR, USA
Copyright 2015 ACM 978-1-4503-3351-1/15/03 ...$15.00
http://dx.doi.org/10.1145/2713168.2713173.

is heavily loaded with traffic from multiple contending stations, and the available bandwidth shared by all stations is not enough to accommodate all incoming traffic, the network is congested. If congestion persists, packet losses due to buffer overflow will occur. In a congested network with many active contending stations, collision induced packet losses may also increase significantly. Packet losses due to these reasons are classified as congestion-caused losses.

In this paper, we exploit the fact that packet losses on a Wi-Fi link can be inferred by not receiving a positive acknowledgement (ACK) packet after reaching the retry limit. We propose that if the packet loss is channel-caused, the MAC layer grants more transmit opportunities by temporarily increasing the retry limit. If the packet loss is congestion-caused, the MAC layer does nothing in order not to conceal the packet loss from higher layer's congestion control algorithms. Our approach implicitly assist existing congestion control mechanisms at higher layers. The dominant transport layer protocols are TCP and UDP. TCP is known for using congestion control. Video traffic accounts for a significant share of UDP traffic and congestion control at the RTP layer is recommended [17] and generally implemented in the newer video telephony systems such as WebRTC [20]. For widely used higher-layer congestion control mechanisms, packet losses are interpreted as indications of congestion which is not always correct especially in a wireless environment, and this may lead to unnecessary reduction in the data sending rate. To mitigate this problem, it is important to reduce channel-caused losses, which requires reliable differentiation between channel-caused losses and congestion-caused losses which in turn necessitates another task of this paper: congestion detection in a Wi-Fi network.

Congestion detection in wireless networks has been extensively studied [1]. However, most of the proposed protocols make use of the detection results in order to perform congestion control [2][3][4][5][6][7][8] or rate adaptation [9][12] at the MAC layer. In [10], congestion status is part of the objective function for optimizing Packet-level FEC (PFEC) and packet scheduling. In [11], the TCP sender generates and sends a special Resource Discovery (RD) packet which travels a round trip and brings back information on the end-to-end capacity to help the sender adjust the sending window size accordingly. In [13], the authors propose to adapt the backoff window size to the current network contention level. In [14], a joint adaptation of link rate and backoff contention window is proposed to improve the performance of 802.11 multi-rate network. In contrast to the aforementioned prior work, in this paper we leave the job of congestion control up to upper layers, like the transport layer or the RTP layer, and make use of the detection results in a very different way, which is adapting the retry limit to the congestion level in the wireless channel to indirectly assist congestion control mechanisms at higher layers.

The retry limit optimization has also been extensively studied in the literature. Representative work includes [16] [26] [27] [28] [29] [30]. In [28], the authors propose adjusting the retry limit according to the MAC layer data rate. However, in Wi-Fi a higher data rate does not necessarily indicate better channel quality which often implies low likelihood of channel-caused losses because hidden terminal interference may still exist or the MAC layer rate adaptation algorithm may try a higher data rate that the current channel condition cannot sustain in order to potentially improve the overall throughput. Even though these mechanisms are shown to be effective since they require either major modifications in the already well-established IEEE 802.11 standard or cross-layer signaling, their applicability is limited. Moreover, most of those prior work uses simulation to validate the proposed solutions. Our solution is implemented in a real testbench that allows for a more realistic evaluation and demonstration.

The main contributions of this paper are:

- We propose a real-time, light-weight and passive congestion detection algorithm which relates the excess data rate with the available MAC-layer capacity and uses only local information within the MAC layer.
- We propose a mathematical expression to quantitatively calculate the congestion level and propose an analytical model to gain valuable insights.
- We design and implement our algorithms in a real WebRTC-based testbench, allowing us to carry out a more realistic evaluation and to demonstrate the practicality.
- Experimental results confirm that our approach can significantly improve the receiver side's video quality of experience by dramatically reducing video freezes or increasing the video bit rate.
- Besides RTP-layer congestion control based applications, our scheme may also help TCP based applications.

The rest of the paper is organized as follows. In Section 2 we introduce the problems confronting WebRTC-based video telephony through experiments and explain the motivation of our work. We discuss the design and implementation details of our solution in Section 3. We also propose an analytical model in Section 3 to gain insight of our proposed scheme. In Section 4 we present our prototype testbench and evaluate our proposed scheme. Finally, we conclude in Section 5.

2. PROBLEM ANALYSIS AND MOTIVATION

We develop a testbench shown in Fig. 10 to perform experiments and identify the problems that we are about to solve.

2.1 Background of WebRTC

WebRTC uses the Google Congestion Control (GCC) algorithm [32] to perform congestion control, which is composed of two parts: the receiver-side controller computes the rate A_r and sends it to the sender; the sender-side controller computes the target sending bit rate A_s that cannot exceed A_r. Specifically the sender-side controller updates the maximum allowable sending rate $A_s(t_k)$ every time t_k the k-th RTCP report message carrying a fraction of lost packets $f_l(t_k)$ arrives at the sender. Usually but not always, the RTCP report message also includes a Receiver Estimated Maximum Bitrate (REMB) message carrying an REMB value A_r. Note that the fraction of lost packets $f_l(t_k)$ is calculated before FEC recovery is applied (if FEC is turned on). As described in [24], the RTCP reports include the fraction of lost packets $f_l(t_k)$ observed by the receiver, while REMB is based on the average delay jitter calculated by the receiver. The sender uses $f_l(t_k)$ to compute the sending rate $A_s(t_k)$, measured in kbps, according to the following

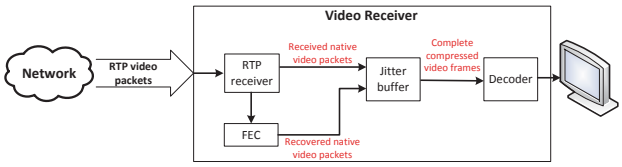


Figure 2: Video bit rate received by the video receiver

equation:

$$A_s(t_k) = \begin{cases} max\{\ X(t_k), A_s(t_{k-1})(1-0.5f_l(t_k))\ \} & f_l(t_k) > 0.1 \\ 1.08\min_{t\in(t_k-\Delta,t_k)} A_s(t) + 1kbps & f_l(t_k) < 0.02 \\ A_s(t_{k-1}) & otherwise \end{cases} \quad (1)$$

where $X(t_k)$ is the TCP friendly rate control (TFRC) rate [25]. The logic behind Eq.1 is: 1) when the fraction of lost packets is considered low ($0.02 \leq f_l(t_k) \leq 0.1$), the data sending rate is kept constant, 2) if the fraction of lost packets is considered high ($f_l(t_k) > 0.1$), the data sending rate is multiplicatively decreased (it is configured that the rate will not be decreased more than once in the last (0.3 + RTT) seconds, where RTT is the round trip time reported by the receiver), but not below $X(t_k)$, 3) when the fraction lost is considered negligible ($f_l(t_k) < 0.02$), the rate is adjusted to be 108% of the minimal value of A_s in the last Δ second (Δ is pre-configured as 1 by default). After $A_s(t_k)$ is computed from Eq.1, the value of $A_s(t_k)$ is further updated as $A_s(t_k) \leftarrow min(A_s(t_k), the\ last\ received\ A_r)$, to ensure that $A_s(t_k)$ never exceeds the last received value of A_r carried in the REMB message.

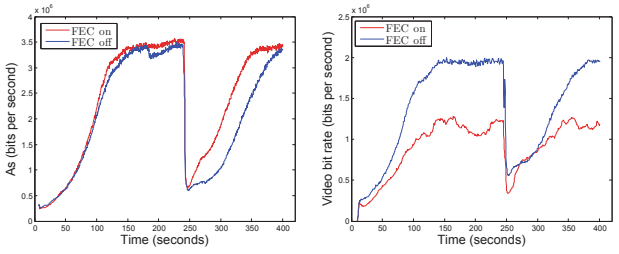
WebRTC utilizes both proactive and reactive packet loss mitigation methods [23]. The proactive method used by WebRTC is packet-level FEC. FEC adds redundancy to achieve packet loss mitigation. In Section 2.3, we will explain the efficiency problem caused by the use of FEC. However, if not using FEC, the reactive packet loss mitigation method alone is not sufficient to mitigate packet losses. In Section 2.4, we will explain what is the reactive packet loss mitigation method used in WebRTC and why it is not good enough. Next we introduce the motivation of our proposed scheme as a better solution in Section 2.5.

2.2 Experimental setup

In this paper, all experimental results are collected by running emulations with the testbench and experimental setup described in Section 4.1 and 4.2 and are the average of 10 emulation runs so that we can do a fair comparison on the experimental results before and after using our proposed scheme.

2.3 Lower received video bit rate due to packet losses

In WebRTC, the application-layer FEC adapts the redundancy to the packet loss rate. Given the same target sending bit rate A_s calculated in Section 2.1, the higher the fraction of lost packets $f_l(t_k)$ reported, the higher the portion of A_s will be assigned to transmit FEC redundant packets, leaving a smaller portion of A_s for sending native video packets and consequently a lower received video bit rate, hence a lower video quality. As shown in Fig. 2, we define the received video bit rate as the arrival bit rate to the video decoder,



(a) (b)

Figure 3: When FEC is turned On and OFF: (a) Target sending rate at Laptop A (b) Received video bit rate at Laptop B

which equals to the total size of complete compressed video frames received by the decoder per second. Due to packet losses in the end-to-end network, the video receiver relies on FEC to recover missing native video packets and fill the gap in the Jitter buffer. When all packets of a video frame arrive at the Jitter buffer, the video frame is sent to the decoder.

By following Section 2.2, we do experiments with FEC turned on or off, and get results in Fig. 3, where (a) shows the target bit rate A_s calculated at the video sender and (b) shows the received video bit rate at the video receiver. From Fig. 3, we can observe that:

a. The maximum value of As is around 3.5 Mbps, while the maximum value of received video bit rate is only around 2Mbps, even when FEC is turned off. This is because WebRTC sets a upper limit which is 2 Mbps for the video encoder, no matter how big As is.

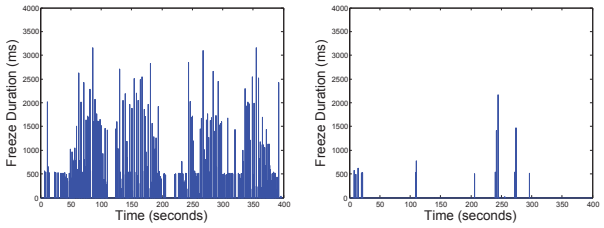
b. At the time of 240 seconds when congestion happens, congestion control algorithms works well and As drops to 0.6 Mbps in both curves.

c. Except the two transient time periods, [0 sec, 100 sec] and [240 sec, 350 sec], although the sending rate As are almost the same, the received video bit rate when FEC is turned off is 70% higher than the case when FEC is turned on, showing significant overhead of FEC.

Due to the efficiency problem explained above, extra complexity and delay of FEC, in some commercialized WebRTC products, like Chrome browser, FEC in WebRTC is disabled. In the next Section 2.4, we will show that however, turning off FEC will bring about another problem.

2.4 Video freezes due to packet losses

As explained in Section 2.1, WebRTC uses both proactive and reactive packet loss mitigation methods. The proactive method is application-level FEC. The reactive approach is based on an end-to-end feedback to request RTP layer retransmissions from the video sender, which is ineffective in the case of large round-trip-time (RTT). In our testbench setup where there is a 300 ms Internet delay, when packet loss happens on the Wi-Fi link from STA_A to AP, the receiver has to wait 600 ms (RTT may range from 50 ms up to 700 ms [31]) before a retransmission from the sender can be received, and this will make the playout buffer delay insufficient to conceal the packet loss. As a result, with the default error concealment technique of WebRTC, a type of frame copy where the video freezes at the last perfectly recovered frame until the lost frame is recovered, excessive video freezes will appear in the decoded video.



(a) (b)

Figure 4: Video freeze duration on Laptop B: (a) FEC is turned off (b) FEC is turned on

Fig. 4 shows the video freezes in a single experiment when FEC is turned off or on, where the freeze duration is calculated by subtracting the time when the next frame is actually displayed from the original scheduled time of the next frame. A significant percentage of the video freezes are close to 600 ms (not exactly 600 ms due to the dynamic value of the playout buffer delay) as shown in Fig. 4 because application layer retransmissions (the reactive approach described above) rely on end-to-end feedback which takes about an RTT (600ms in the experiment). If application layer retransmissions also get lost, the actual video freezes can be multiple times of one RTT. Although the use of FEC can significantly reduce the video freezes, as shown earlier, it can significantly reduce the received video bit rate.

2.5 Motivation

In summary, channel-caused packet losses on an edge Wi-Fi link at the video sender (such as the link from STA_A to AP in Fig. 10) poses a dilemma to WebRTC. If FEC is turned on, the almost 70% overhead can lead to lower video quality at the video receiver; if FEC is turned off, like some commercial products do, excessive video freezes would happen at the video receiver if the RTT is not negligible.

Supposing the differentiation between channel-caused losses and congestion-caused losses has already been perfectly done (we will present an imperfect but effective algorithm in Section 3.1), there are two candidate solutions based on our analysis:

1. (cross-layer early feedback) The MAC layer of the video sender detects a packet loss by not successfully receiving a positive acknowledgement within the maximum allowed number of retransmissions. On behalf of the video receiver, inside the video sender the MAC layer sends a spoofed negative acknowledgement(NACK) to RTP layer to trigger an RTP layer retransmission in time, which effectively reduces the RTT and makes the receiver possibly unaffected by the packet loss.
2. (local only approach) the MAC layer adaptation grants the packet which will experience an imminent loss (i.e., the packet is about to be discarded and considered as lost by a higher-layer protocol after reaching the retry limit) higher priority or more opportunities to prevent the loss.

The first solution relies on deep packet inspection of the lost packet in order to find the associated sequence number so that the spoofed NACK can include the sequence number and tell the video sender which packet gets lost. However, if Transport Layer Security (TLS) is used at the application layer for security, deep packet inspection becomes essentially impossible because TLS encrypts the entire payload. Moreover, to our best knowledge Secure Real-time Transport Protocol (SRTP) profile is enabled by default in WebRTC. Every incoming packet has to be successfully decrypted before being further processed. Since the encryption key is hard to get at the MAC layer, the MAC layer cannot generate an encrypted NACK that is understandable to the RTP sender, making the cross-layer feedback infeasible.

The second solution, unlike EDCA in 802.11e, which depends on the type of the packets (e.g., voice vs. video) and grant different priorities to different Access Categories, ignores the type of the packets and treats every packet equally to avoid relying on cross layer assistance. The rationale is that in the case of non-congestion, network resource anyway is under-utilized, granting more transmission opportunities only utilizes otherwise wasted network bandwidth. This solution is the one we use in this paper. There are still some challenges, like how to guarantee fairness in contention, how to avoid incurring congestion if granting too many opportunities, and how to make sure higher priority or more opportunities indeed prevent loss. We will answer these questions partly as follows and partly in Section 3.2.

Basically, we have two paramters to do MAC layer adaptation: Retry limit and contention window (CW). The duration of CW is used for resolving contention when several stations are competing to access the same channel. So a change in the CW of a STA means a change in the medium access priority, which affects fairness and is undesirable for other stations that do not use our mechanism. In contrast, changing the retry limit does not affect each single contention and we can still maintain equal channel access success probabilities. In a lightly loaded network, increased retransmissions can better utilize the network resources; in a heavily loaded network, the packet loss is classified as contention-caused loss, and retry limit will not be increased. Regarding change in the retry limit, there are open challenges, which will be discussed in section 3.2.

3. SYSTEM DESIGN AND IMPLEMENTATION

3.1 Congestion detection

In IEEE 802.11 wireless networks, congestion may be defined as a state where the shared wireless medium is close to being fully utilized by the stations, under certain channel conditions and/or external interference [18]. Unlike wired networks, where throughput degradation is indicative of congestion, in wireless networks throughput degradation can occur due to a lossy channel, increased packet collisions during congestion or external interference. In addition, throughput of a wireless link is also directly influenced by the rate adaptation algorithm through its choice of the transmission data rate. If a lower data rate is in use, the throughput for a given time interval will be lower than that with a higher data rate.

For these reasons, several studies have proposed the use of medium utilization as a measure of congestion in the wireless medium [18] [19], where the authors show that medium utilization can be used to classify network state as uncongested, moderately congested, and highly congested. However, it is not easy to find a threshold value of channel uti-

lization to clearly identify congested and uncongested cases. Also, obtaining an accurate channel utilization from real-time measurements is a challenge. So in this paper we propose another metric other than channel utilization to detect congestion.

In this paper, we design, implement and evaluate a real-time, light-weight and passive congestion detection technique along with a retry limit adaptation algorithm for Wi-Fi networks based on only local information.

3.1.1 Congestion metric

For each Wi-Fi station, a single queue is typically used for traffic with the same priority (called an access category in 802.11e), and the queue length at any time instant t_0 is given by:

$$Q_{len}(t_0) = \int_0^{t_0} (Ar(t) - Deliv(t) - Disc(t))\,\mathrm{d}t \qquad (2)$$

where $Ar(t)$ is the data arrival rate from the IP layer at time t; $Deliv(t)$ is the delivery rate (number of successfully delivered bits per unit time), determined by the available shared bandwidth; $Disc(t)$ is the discard rate (due to reaching the retry limit), determined by the retry limit, the random backoff time and the channel occupancy by contending peers.

From a quantitative perspective, a congested network is defined as one of which the aggregate data arrival rate is persistently higher than the network capacity. Similarly, a congested Wi-Fi station can be defined as one of which the data arrival rate is more than the share of bandwidth available to that station, i.e., $\int_0^{t_0}(Ar(t) - Deliv(t))dt$ is greater than a positive threshold. From Eq.(2), we note that the queue length does not fully characterize congestion, as a successfully delivered packet is treated the same way as a discarded packet.

Now we consider how to get the average transmit delay of a discarded MPDU, say, TD, and then use its reciprocal $1/TD$ to obtain an upper bound on $Disc(t)$. Here the transmit delay is defined to be the time interval from the time the MPDU reaching the head of its MAC queue for transmission, to the time an acknowledgement for this packet is received or discarded upon reaching its retry limit. In other words, the queueing delay due to waiting for the service of previous packets to be completed is not included. As specified in the 802.11 standard, upon a packet loss, the Wi-Fi station randomly chooses a backoff time and retransmits the packet after the backoff time expires. However, other contending stations may occupy the channel during the backoff time and force the backoff timer to freeze until the channel is sensed idle again after a DIFS/AIFS period. Therefore, the actual length of the backoff period can be much longer than the original randomly picked backoff time. Borrowing the concept of conditional collision probability and the equation from [15] and [16], and assuming that the channel is occupied by other contending stations with a constant and independent probability p at each time slot, it can be shown that the average transmit delay of a discarded MPDU is:

$$\overline{TD(R,L)} = \sum_{i=1}^{R}\left(\frac{min(2^{i-1}\cdot(CW_{min}+1)-1, CW_{max})}{2} \times(p\cdot T(L) + aSlotTime) + T(L)\right) \qquad (3)$$

where $T(L_i) = T_{DATA}(L_i) + aSIFSTime + T_{ACK} + aDIFSTime$ is the transmission time for sending an L-byte long packet. p is the conditional collision probability [15]. R is the retry limit with a default value 7 (including the first transmission and maximum 6 retransmissions). L is the length of a packet. Assume that the PHY data rate = 65 Mbps, L = 1224 bytes, ACK = 76 bytes, PLCP overhead = 40 us, aSlotTime = 9 us, aSIFSTime = 16 us, and aDIFSTime = 34 us. Then T(1224) = 0.09 ms + 0.16 ms = 0.25 ms.

In a legacy network with $CW_{min} = 15$ and $CW_{max} = 1023$, assuming $p = 0.1$, we have $\overline{TD(7,1224)} = 36.175ms$. In an EDCA network, for the video AC with $CW_{min} = 7$ and $CW_{max} = 15$, assuming $p = 0.1$, we have $\overline{TD(7,1224)} = 3.399ms$.

Apparently, a packet of the video access category in an EDCA netowrk ($3.399ms$) takes much less time before being discarded than the time ($36.175ms$) taken by a packet in legacy network. As a result, by the upper bound $Disc(t) \leq 1/\overline{TD(R,L)}$ (assuming each discarded packet takes the same average delay), the $Disc(t)$ in Eq.(2) could take a larger value that is non-negligible in an EDCA network. The observation inspired us to find a better congestion metric other than the queue length to make our mechanism work in a broad range of networks. To accommodate this purpose, we define the excess data rate as follows:

$$EDR_\tau^w(t) \leftarrow [Ar_\tau^w(t) - Deliv_\tau^w(t)]^+/(w\tau) \qquad (4)$$

where $[x]^+ = max(x, 0)$. $Ar_\tau^w(t)$ is the total amount of arrival data (aggregate size of arrival MSDUs) in bits and $Deliv_\tau^w(t)$is the total amount of successfully delivered data (aggregate size of successful MPDUs) in bits during the time period $[t - w\tau, t)$. τ is the sampling interval. To reduce short-term fluctuation, a sliding window w is introduced, which represents the number of time intervals (each of τ long) so that all statistics falling in and only in the time period $[t - w\tau, t)$ contribute to the calculation in the Eq.(4). In other words, τ determines the update granularity, and w determines the update smoothness. For example, if $\tau = 0.1$, $w = 10$, $EDR_{0.1}^{10}(t)$ stands for the average excessive data rate (measured in bits per second) during the most recent $0.1 \times 10 = 1$ second.

Given the same excess data rate, different networks may experience different levels of congestion because the available bandwidth may be different. To manifest the severity of congestion, we divide excess data rate by the estimated MAC layer capacity to get the congestion level

$$CL(t) = EDR_\tau^w(t)/MC_\tau^w(t) \qquad (5)$$

where the estimated MAC layer capacity $MC_\tau^w(t)$ is given by

$$MC_\tau^w(t) = Deliv_\tau^w(t)\Big/\sum_{i=1}^{N} TD(r_i, L_i) \qquad (6)$$

Where $TD(r_i, L_i)$ is the actually measured transmit delay for the i-th transmitted packet. i stands for the i-th packet and r_i is the number of transmissions (including the first transmission and the subsequent retransmissions) that the i-th packet performes. L_i is the packet length of the i-th packet. The rationale behind Eq.(6) is: assuming during the most recent time window $(t - w\tau, t)$, a station transmits N fresh (excluding retransmissions) packets, the available

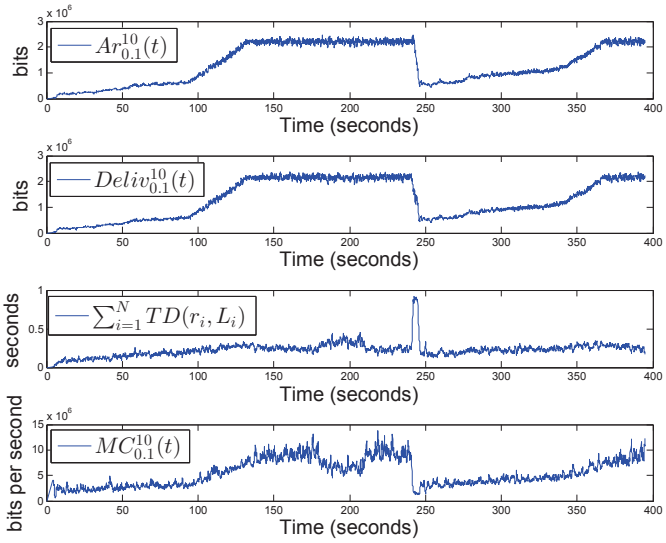


Figure 5: Realtime calculations of Eq.(6) in WebRTC-based video teleconferencing experiments

MAC layer capacity for this station is estimated as the total successfully delivered data (measured in bits) divided by the total amount of transmit delay experienced.

3.1.2 Evaluation of the congestion metric

We propose to calculate the value of Eq.(5) in realtime to detect the congestion level in the network. As shown in Eq.(5), the ratio of $EDR^w_\tau(t)$ to $MC^w_\tau(t)$ determines the congestion metric. We start to analyze the estimated MAC layer capacity in Eq.(6), which is used in Eq.(5).

Fig. 5, from top to bottom, shows the curves for $Ar^{10}_{0.1}(t)$, $Deliv^{10}_{0.1}(t)$, $\sum_{i=1}^{N} TD(r_i, L_i)$ and $MC^{10}_{0.1}(t)$ respectively. Two interesting observations of the curve of $MC^{10}_{0.1}(t)$ can be noticed:

a. Large fluctuations of $MC^{10}_{0.1}(t)$ at the very beginning.
b. $MC^{10}_{0.1}(t)$, although fluctuating over small time scales, follows a trend of increasing when the traffic arrival rate increases in the time interval [0, 140 sec] during which there is no cross traffic.

The reason for observation **a** is that the sample size is too small(small number of packets, and very short time duration) at the beginning, and we will explain later that this will not affect the correctness of Eq.(5). We explain observation **b** using the theorem shown below. Since an RTP layer video packet is typically much bigger than the Maximum Transmission Unit (MTU) size of the Ethernet, it is usually fragmented into multiple IP packets, all of which have approximately the same size as the MTU size, except the last one which carries the remainder. Thus, in the theorem below we assume that every packet at the MAC layer has the same length.

THEOREM 1. (MAC layer capacity estimation) *Given that every packet has the same length, i.e., $L_i = L$, for all i, $TD(R, L) > \tau_b$, where R is the retry limit and τ_b is the busy time interval of the hidden terminal traffic as shown in Fig. 6, in a Wi-Fi network with no competing traffic, we have:*

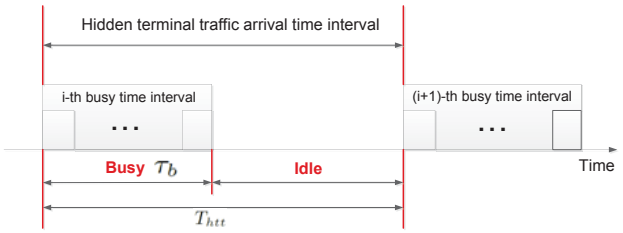


Figure 6: Network resource occupation of hidden terminal traffic

(i) $MC^w_\tau(t)$ monotonically increasing with $Ar(t)$ but slower than $Ar(t)$;
(ii) $MC^w_\tau(t)$ is bounded as follows:

$$\big(Ar(t)/(Ar(t)+\rho C)\big)C \leqslant MC^w_\tau(t) \leqslant C \tag{7}$$

where ρ is the channel utilization of the hidden terminal, and C is the MAC layer capacity if there is no hidden terminal traffic.
(iii) if further assuming that $Ar(t)$ follows the Poisson traffic model [33], then

$$MC^w_\tau(t) = \Big(\lambda L/\big(\lambda L + \rho C(\lambda\tau_b + e^{-\lambda\tau_b} - 1)/\lambda\tau_b\big)\Big)C \tag{8}$$

where λ is the packet arrival rate in the Poisson traffic model.

PROOF. **Part (i):** Since there is no cross traffic, collision with hidden terminal traffic is the only reason for transmission failure. The assumption that $TD(R, L) > \tau_b$ means that the transmit delay of one packet is so big that all subsequent packets are forced to be transmitted in an idle time interval as shown in Fig. 6. Therefore there is at most one packet that will be transmitted during a busy time interval of the hidden terminal within each hidden terminal traffic arrival time interval T_{htt} (in Fig. 6). Assuming $M(M \geqslant 1)$ fresh MPDUs (excluding retransmissions) are transmitted over the network within one T_{htt}, the aggregate transmit delay is given by

$$\sum_{i=1}^{M} TD(r_i, L) = \sum_{i=2}^{M} TD(1, L) + TD(r_1, L) \tag{9}$$

where $TD(r_1, L)$ means that the first fresh MPDU is possibly transmitted during a busy time interval, and the number of retry count is r_1 ($r_1 \in [1, 7]$, assume retry limit is 7.) which depends on at which time instant the packet arrives in a busy time period. Each of the other $(M-1)$ packets which must arrive during an idle time period, only needs to be transmitted once ($TD(1, L)$), where 1 stands for only one transmission will be taken for each one of the $(M-1)$ packets. Since $TD(r_1, L)$ is usually much bigger than $TD(1, L)$ due to the exponential growth of the contention window (CW) of successive retransmissions, $\sum_{i=1}^{M} TD(r_i, L)$ would increase as M increases but at a pace slower than M. Now going back to Eq.(6), as all M packets gets delivered (the first one may be delayed but finally falls within an idle interval and also goes through), the total delivered bits $Deliv(t)$ grow at the same speed of M. By Eq.(6), the $MC^w_\tau(t)$ monotonically increases but at a pace (in percentage per unit time) slower than $Ar(t)$ does, where $Ar(t)$ is the total amount of data in the M packets.

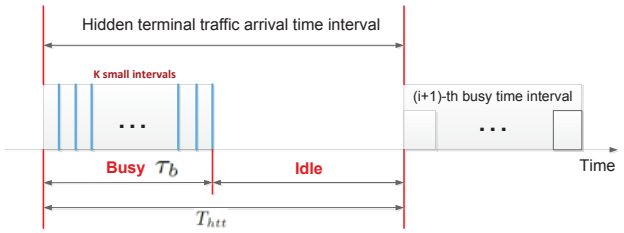


Figure 7: Segmented network resource occupation of hidden terminal traffic

Part (ii): Considering a time interval $(t - w\tau, t)$ longer than a single T_{htt} at any time t, if there are N fresh MPDUs of data traffic and Q hidden terminal traffic arrival time intervals $(Q \times T_{htt})$, where $Q = \frac{w\tau}{T_{htt}}$ and is an average value, then there will be at most Q fresh MPDUs colliding with the hidden terminal traffic and each will experience a transmit delay not longer than τ_b. The total transmit delay $\sum_{i=1}^{N} TD(r_i, L)$ satisfies

$$\sum_{i=1}^{N} TD(1, L) \leqslant \sum_{i=1}^{N} TD(r_i, L) \leqslant \sum_{i=Q+1}^{N} TD(1, L) + Q\tau_b \quad (10)$$

Since $\sum_{i=1}^{N} TD(1, L) = Ar_\tau^w(t)/C$ and $\rho = \tau_b/T_{htt}$, Eq.(10) can be relaxed and simplified to

$$Ar_\tau^w(t)/C \leqslant \sum_{i=1}^{N} TD(r_i, L) \leqslant Ar_\tau^w(t)/C + w\tau\rho \quad (11)$$

Considering Eq.(11) and Eq.(6), and noting that $Ar_\tau^w(t) = Deliv_\tau^w(t)$ when there is no loss, we can get

$$\big(Ar_\tau^w(t)/(Ar_\tau^w(t) + w\tau\rho C)\big)C \leqslant MC_\tau^w(t) \leqslant C \quad (12)$$

Inside Eq.(12), $Ar_\tau^w(t)$ is the total number of bits arriving during time duration $(t - w\tau, t)$ at an average arriving rate $Ar(t)$, so by replacing $Ar_\tau^w(t)$ with $w\tau Ar(t)$, we get Eq.(7). When the arrival rate is very small($Ar(t) \ll \rho C$), we get the lower bound $Ar(t)/\rho$, and when the arrival rate is large $(Ar(t) \gg \rho C)$, the upper bound gets closer to C and the range in Eq.(7) becomes tighter.

Part (iii): As we explained earlier, within each hidden terminal traffic arrival time interval T_{htt}, only the transmit delay of the first packet possibly gets affected, so we calculate the expected delay of each first packet. As shown in Fig. 7, the busy time interval is divided into K small intervals, each of equal length τ_b/K. If there is at least 1 packet arriving during the first time interval, then the delay will be approximately $(1 - 1/K)\tau_b$ while the probability that this event happens is $Prob(\boldsymbol{N_1} \geqslant 1)$, where the random variable $\boldsymbol{N_1}$ means the number of packets that arrive in the first small interval. If there is at least 1 packet arriving in the second time interval but there is no packet arriving before the second time interval, then the delay will be approximately $(1 - 2/K)\tau_b$ while the probability that this event happens is $Prob(\boldsymbol{N_2} \geqslant 1)Prob(\boldsymbol{N_1} = 0)$. Similarly, if there is at least 1 packet arriving in the i-th interval but there is no packet arriving before the i-th interval, then the delay will be approximately $(1 - i/K)\tau_b$ while the probability that this event happens is given by

$$Prob(\boldsymbol{N_i} \geqslant 1)Prob(\sum_{j=1}^{i-1} \boldsymbol{N_j} = 0) \quad (13)$$

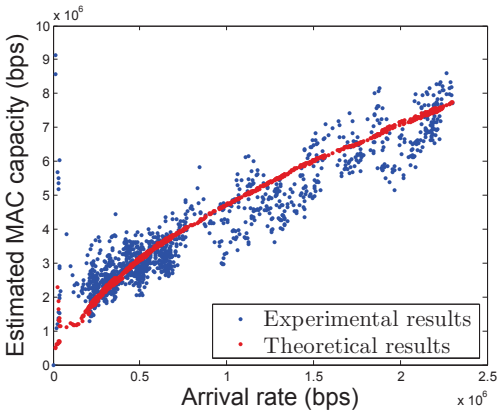


Figure 8: Comparison between theoretical results and experimental results of estimated MAC layer capacity as a function of the arrival rate

The average delay of these first packets is given by

$$\begin{aligned} Delay_K^{\tau_b} &= \sum_{i=1}^{K-1} \Big((1 - i/K)\tau_b Prob(\boldsymbol{N_i} \geqslant 1) \\ &\quad \times Prob(\sum_{j=1}^{i-1} \boldsymbol{N_j} = 0) \Big) \\ &= \sum_{i=1}^{K-1} \Big((1 - i/K)\tau_b (1 - e^{-\lambda\tau_b/K}) e^{-(i-1)\lambda\tau_b/K} \Big) \\ &= (1 - 1/K)\tau_b \\ &\quad - \tau_b e^{-\lambda\tau_b/K} (1 - e^{-\lambda\tau_b(K-1)/K}) / \big(K(1 - e^{-\lambda\tau_b/K})\big) \end{aligned} \quad (14)$$

where λ is the packet arrival rate in the Poisson traffic model. If there is no packet arriving before the K-th interval, then the delay $(1 - K/K)\tau_b$ will be zero. And this is why the summation excludes K in Eq.(14). As $K \to \infty$, our approximation becomes arbitrarily accurate, and by the L'Hopital's Rule we get the actual average transmit delay,

$$Delay^{\tau_b} = \lim_{K \to \infty} Delay_K^{\tau_b} = \tau_b - (1 - e^{-\lambda\tau_b})/\lambda \quad (15)$$

Considering $\sum_{i=1}^{N} TD(1, L) = Ar_\tau^w(t)/C$, $\rho = \tau_b/T_{htt}$ and $Q = \frac{w\tau}{T_{htt}}$, now we can rewrite the total transmit delay $\sum_{i=1}^{N} TD(r_i, L)$ as

$$\begin{aligned} \sum_{i=1}^{N} TD(r_i, L) &= \sum_{i=1}^{N} TD(1, L) + Q Delay^{\tau_b} \\ &= Ar_\tau^w(t)/C + w\tau\rho(\lambda\tau_b + e^{-\lambda\tau_b} - 1)/\lambda\tau_b \end{aligned} \quad (16)$$

Substituting Eq.(16) into Eq.(6) and noting $Ar_\tau^w(t) = w\tau Ar(t) = w\tau\lambda L$ (since $Ar(t) = \lambda L$), we get Eq.(8). □

In the above proof, we use the technique of subdividing a time interval into K smaller ones, and then taking the limit $K \to \infty$. Similar techniques have been used elsewhere, for example, [34, p. 7].

We now look at whether our experimental results match the above theorem. In the experiment, the hidden terminal traffic consists of only UDP-based video packets, each of the same length (1224 bytes) at the MAC layer, with

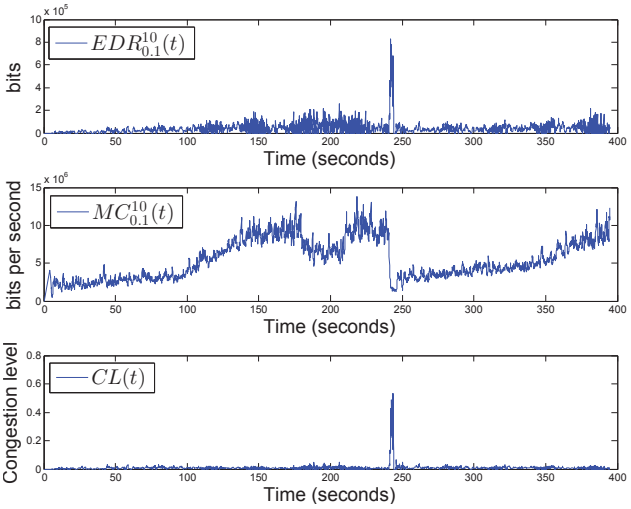


Figure 9: Realtime calculations of Eq.(5) in WebRTC-based video teleconferencing experiments

constant frame rate and constant frame size. So the T_{htt} and τ_b in Fig. 7 are constant and take values $0.033s$ and $0.01075s$, respectively. According to Eq.(3), a packet discarded after the retry limit takes $TD(7, 1224) = 10.894ms$ ($R = 7$ and $L = 1224$), where $p = 0$ is used because the interference from the hidden terminal traffic cannot be heard and the backoff timer will not freeze. Since $\tau_b = 0.01075s < 10.894ms$, the condition in Theorem 1, $TD(R, L) > \tau_b$, is satisfied. However, packets from WebRTC do not have the same length because audio packets usually are much shorter than video packets, which does not meet the assumption that every packet has the same length L in Theorem 1. So we use the average packet size as the value for L. As the sending rate As gradually increases according to Eq.(1), the average packet size is expected to increase as well because video packets account for an increasing portion of the total packets. Since MAC layer capacity C in the theorem refers to the MSDU throughput, i.e., $(L/(L/DataRate+ACK/DataRate+PLCP_overhead+aSIFSTime+aDIFSTime))$, where the $DataRate$ is the PHY data rate and the other parameters are fixed and use the same value as in section 3.1.1, the value of C dynamically changes with the value of average packet size L.

The explanation above allows us to draw the theoretically estimated MAC layer capacity as a function of arrival rate according to Eq.(8), and compare the theoretical results with the experimental results in Fig. 8, where we only take the data falling in the time window [0, 140 sec] of Fig. 5 because there is no competing traffic during this time window. As Fig. 8 shows, the two results match well, even though WebRTC traffic is not memoryless and does not follow the Poisson traffic model. Note that for the same value of the arrival rate, the estimated MAC layer capacity may be different because the total number of packets and averaged packet size may be different.

Fig. 9 shows the congestion level $CL(t)$ as a function of time t, which is the ratio of the $EDR_{0.1}^{10}(t)$ shown in Fig. 9 to the estimate of the MAC layer capacity $MC_{0.1}^{10}(t)$ also shown in Fig. 9. Before congestion happens and the network is lightly loaded, the estimate of MAC layer capacity does not accurately manifest the true MAC layer capacity. But this inaccuracy does not lead to inaccurate congestion detection because the excess data rate $EDR_{0.1}^{10}(t)$ is zero most of the time, and the congestion level $CL(t)$ is close to zero regardless of the value of the estimate of the MAC layer capacity. At time 240 seconds, when congestion happens, the estimate of the MAC layer capacity sharply decreases and the excess data rate increases dramatically, resulting in a clear jump in the value of congestion level $CL(t)$. This significant jump allows an easy threshold to be set to do congestion detection.

3.2 Challenges in adjusting the Retry Limit

It is important to find a good compromise for value of the the retry limit, because:

1. The random backoff period in a noisy channel or a heavily loaded channel can be quite different (due to backoff timer freeze), so excessive retransmissions in heavily loaded channel make performance worse. According to Eq.(3), in a lightly loaded channel, assuming $p = 0$, we have $\overline{TD(7, 1224)} = 10.894ms$; in a heavily loaded channel,assuming $p = 0.9$, we have $\overline{TD(7, 1224)} = 1.75ms + 236.93ms = 238.68ms$. If a packet takes 238.68 ms to transmit, then the transmit time for a video frame (which usually includes multiple packets) can be hundreds of milliseconds, which is much longer than a typical inter-frame interval. Even worse, delay in transmitting the current packets will introduce delays to subsequent packets, causing more and more delay, that is, the delay is cumulative.
2. Excessive retransmissions also reduce the drainage speed of the MAC layer buffer and increase the probability of buffer overflow.
3. Excessive retransmissions also make a packet's delay larger such that the packet may miss the decoding deadline at the receiver, making all retransmissions of that packet a waste of network resources.
4. Insufficient retransmissions not just add extra traffic at the MAC layer but also leave the packet loss problem unresolved.

In our design, the retry limit is not increased in the case of congestion, so problem 1 does not happen. Our approach checks the value of $CL(t)$ in Eq.(5) before performing each extra retransmission. If the buffer gradually builds up, this means that the value of excess data rate in Eq.(4) is consistently positive while the output of Eq.(5) remains below the predefined threshold. In this case, our algorithm checks the available buffer size to avoid buffer overflow so problem 2 does not happen. Regarding 3 and 4, we will show in section 4 that they rarely happen.

In this paper, we propose that in the case where a channel-caused packet loss happens, the value of the retry limit is raised so that granting more retransmissions to that packet can bring significant performance gain. However, finding an optimal value of the retry limit is not our focus(left for future work).

3.3 Congestion aware MAC layer adaptation algorithm

We now present the details of the algorithms discussed earlier. Algorithm 1 runs periodically in the background to collect the total amount of arrived data, the total amount

Algorithm 1: UpdatePacketsStatistics()

// This function runs periodically every τ seconds. Default value for τ is 0.1.
Input: $TotalBitsFromIP$, $TotalBitsSucess$ and $TotalTXDelay$
Output: Updated lists: $ListAr\{\}$, $ListServ\{\}$, $ListTXDelay\{\}$
// w is the window size. Default value of w is 10. Replace the oldest element with the new one
if $ListAr.size() \geqslant w$ **then**
    $ListAr.pop\_back()$; // Delete the oldest element
    $ListServ.pop\_back()$;
    $ListTXDelay.pop\_back()$;
end
$ListAr.push\_front(TotalBitsFromIP)$; // Insert an element at the beginning
$ListServ.push\_front(TotalBitsSucess)$;
$ListTXDelay.push\_front(TotalTotalTXDelay)$;
$TotalBitsFromIP = 0$; // Reset these value to collect new statistics for the next τ interval
$TotalBitsSucess = 0$;
$TotalTXDelay = 0$;

of delivered data and the total transmit delay within the most recent time interval τ. Algorithm 1 also makes sure that the three lists in Eq.(5) contain the newest data within the time window $w\tau$. Algorithm 2 calculates the current congestion level $CL(t)$ and is called as needed. Algorithm 3 determines whether the retry limit will be increased or not, and it is called before each retransmission. If the MPDU is a retransmission and has reached the default retry limit, the algorithm will check if any of the three conditions is satisfied: the network is congested, the available buffer size is too small or the predefined extended retry limit is about to be exceeded. If yes, this retransmission will be given up; if not, this retransmission will be performed. Note that the predefined extended retry limit is configurable.

Algorithm 2: CalculateCongestionLevel()

Input: $ListAr\{\}$, $ListServ\{\}$, $ListTXDelay\{\}$
Output: $CL(t)$
$Ar^w_\tau(t)$ = Sum of all elements in list $ListAr\{\}$;
$Deliv^w_\tau(t)$ = Sum of all elements in list $ListServ\{\}$;
$\sum_{i=1}^{N} TD(r_i, L)$ = Sum of all elements in list $ListTXDelay\{\}$
$EDR^w_\tau(t) = max\{Ar^w_\tau(t) - Deliv^w_\tau(t), 0\}/(w\tau)$;
$CL(t) = EDR^w_\tau(t) \sum_{i=1}^{N} TD(r_i, L)/Deliv^w_\tau(t)$;
return $CL(t)$;

4. EXPERIMENTAL EVALUATION

In this section, we evaluate how our congestion-aware MAC layer adaptation scheme could improve WebRTC-based video teleconferencing over Wi-Fi. Specifically the goal of our evaluation is to show (a) less fluctuations and decrease in the sending rate at the video sender and (b) less video freezes at the video receiver can be achieved, by applying our adaptation scheme to reduce channel-caused packet losses.

4.1 Testbench setup

As shown in Fig. 10, the testbench consists of three directly connected laptops via Ethernet cable, where Laptop A and Laptop B run WebRTC (version 6475) [20] based video teleconferencing while the third laptop called the OPNET Laptop runs OPNET with system-in-the-loop (SITL) [21] functionality. SITL allows real WebRTC packets (including RTP packets and RTCP packets) to enter the OPNET Laptop from Laptop A and Laptop B. In this way, we can emulate the delivery of these packets through various communication networks with high fidelity.

Algorithm 3: MACLayerTransmitMPDU()

...
if *this is a retransmition* **then**
    // r is retry count with initial value 0. R is the default retry limit.
    if $r \geqslant R$ **then**
        // CL_{th} is the threshold for determining congestion status. BF_{th} is the buffer size threshold. R_{ex} is retry limit extension.
        if $CalculateCongestionLevel() \geqslant CL_{th}$ *OR*
        Current Buffer size $\geqslant BF_{th}$ *OR*
        $r \geqslant (R + R_{ex})$ **then**
            Delete(MPDU) ; // Network is congested, or the available buffer size is too low, or the extended Retry Limit will be exceeded.
            return ;
        else
            $r = r + 1$; // update retry count value.
            if $r == (R + 1)$ **then**
                reset CW ; // Reset the contention window so that subsequent extended retransmissions would be like belonging to a new fresh MPDU.
            end
        end
    end
end
Transmit this MPDU ;
...

Inside the OPNET Laptop of Fig. 10, a typical and realistic network scenario is created, which represents an important lossy communication network that consists of the Internet in the core and Wi-Fi links on the edge. Real WebRTC traffic from Laptop A and B via Ethernet interfaces arrives at SITL_1 and SITL_2 then STA_A and STA_B. We can consider STA_A and STA_B as virtual mapping nodes of Laptop A and B, respectively. Between STA_A and STA_B, a Wi-FI network and the Internet are simulated. A detailed description about the testbench is summarized shortly. Without loss of generality, this paper only investigates packet losses that happen when STA_A sends WebRTC video packets to AP, and implement our algorithms in STA_A to improve the overall performance of the video flow from Laptop A to Lap-

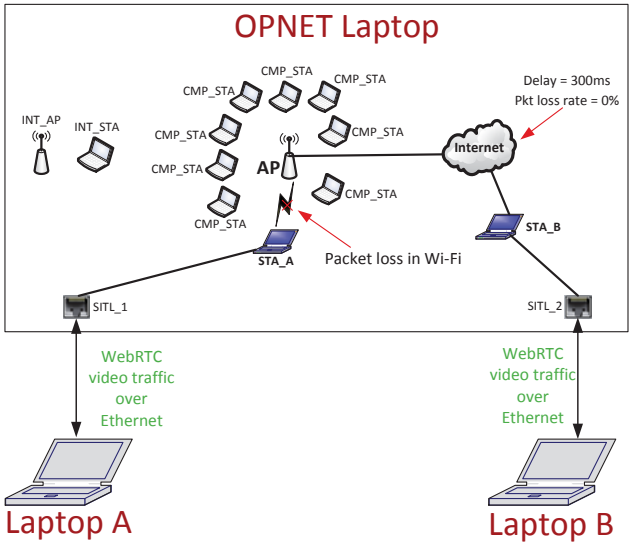


Figure 10: WebRTC generated real video traffic passes through an OPNET-based network emulator

top B. Simply put, our approach improves the video sender who has a lossy Wi-Fi link.

Summary of the testbench:

PHY and MAC layer: HT PHY at 2.4 GHz with IEEE 802.11n. PHY Data Rate is 65 Mbps. All other parameters use default values in OPNET 17.1.A. Since WebRTC video traffic uses Best Effort Access Category by default, we let all other traffic (cross traffic and hidden terminal traffic) in the Testbench use Best Effort as well.

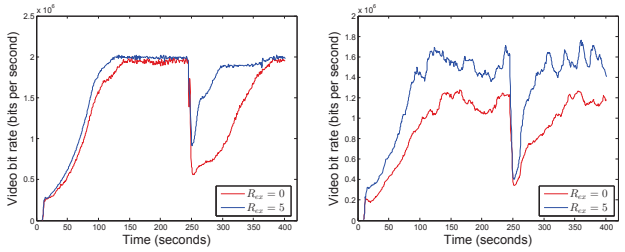
Communication path: Laptop A via Ethernet ↔ SITL_1 ↔ STA_A ↔ AP ↔ Internet ↔ STA_B ↔ SITL_2 ↔ Laptop B via Ethernet.

Main Wi-Fi network: Consists of STA_A, AP and 8 CMP_STA with 802.11n 2.4G Hz. The AP is at a fixed position, while STA_A and CMP_STA are randomly placed (But the distance constraints required to create a hidden terminal network are satisfied).

Congestion: Adjust the cross traffic between CMP_STA and AP to create different levels of congestion.

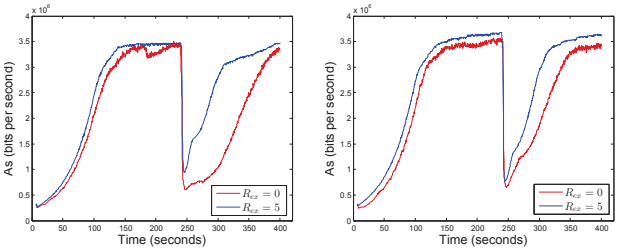
Hidden terminal Wi-Fi network: It consists of INT_STA and INT_AP with fixed locations. INT_STA and STA_A cannot hear each other, but AP is within interference range of INT_STA. Since INT_AP cannot be interfered by anyone in the Main Wi-Fi network, INT_STA servers as a hidden terminal to STA_A but not vice versa.

Traffic definition: All three types of traffic (WebRTC, cross traffic and hidden terminal traffic) are UDP-based video traffic, but some differences are there. Hidden terminal traffic uses constant bit rate after it gets started. Cross traffic also uses constant bit rate for easier control but the rates are different in different time periods in order to create different levels of congestion. WebRTC starts from a minimum target bit rate (50 kbps) and then gradually evolves as explained in Section 2.1. The default maximum target bit rate is around 4 Mbps. The target bit rate goes into the video encoder which generates a final video bit rate generally not higher than the target bit rate. Another difference is that cross traffic and hidden terminal traffic do not perform congestion control but WebRTC uses GCC (refer to Section 2.1 for more detail) for congestion control.



(a) (b)

Figure 11: Received video bit rate on Laptop B: (a) FEC is turned off (b) FEC is turned on. R_{ex} stands for retry limit extension in Algorithm 3. $R_{ex} = 0$ means that our adaptation algorithm is not used. This definition applies to all following figures.



(a) (b)

Figure 12: Target sending rate of Laptop A: (a) FEC is turned off (b) FEC is turned on.

4.2 Experimental setup

We combine different network conditions (hidden terminal traffic, competing traffic with both high and medium load) and carry out a large number of experiments, each lasting 400 seconds. The hidden terminal traffic starts at the beginning (0 second) until the end of each experiment. Competing traffic is active within two time periods, [180 sec, 210 sec] and [240 sec, 250 sec], where the former time period has competing traffic of medium load and the latter has competing traffic of high load.

4.3 Improved video bit rate

As explained in Section 2.1, the video sender of WebRTC uses loss-based FEC. If we could reduce the number of packet losses, we expect to see fewer FEC redundant packets being sent out, which results in a higher video bit rate from the video sender. In Fig. 11 (b) where FEC is turned on, we are able to confirm the expected improvement which is around 40%. In Fig. 11 (a) where FEC is turned off, we can still see some gain because the target bit rate A_s is higher if our approach is used, which is shown in Fig. 12 (a).

4.4 Improved target sending rate

Our approach reduces packet losses so that the fraction of lost packets $f_l(t_k)$ reported is smaller. According to Eq.(1), target bit rate As will be smooth and increase steadily, and this can be seen In Fig. 12 (a) and Fig. 12 (b). When the network is congested, our approach is able to detect the congestion and does not increase the retry limit, so that the fraction of loss $f_l(t_k)$ increases, allowing WebRTC's congestion control algorithm to be aware of the congestion status and properly reduce the value of A_s to mitigate the conges-

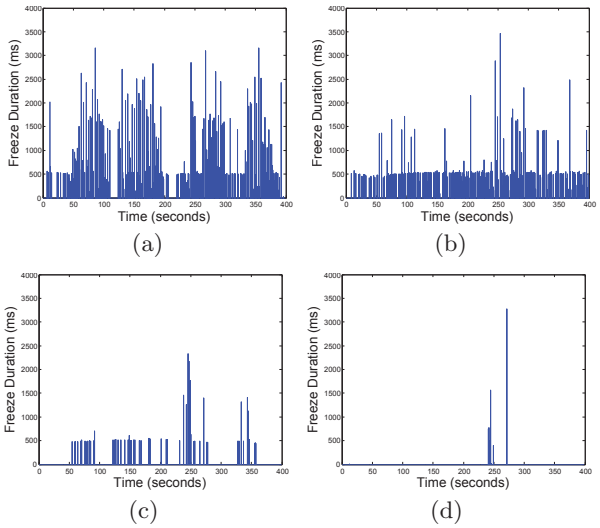


Figure 13: Video freeze duration on Laptop B (FEC is turned off): (a) $R_{ex} = 0$, (b) $R_{ex} = 3$,(c) $R_{ex} = 5$ and (d) $R_{ex} = 7$.

tion, as shown in both Fig. 12 (a) and Fig. 12 (b) at time 240 sec.

4.5 Reduced Video freezes

Fig. 13 shows that the total amount of video freezes is significantly reduced by using our approach when FEC is turned off. When the value of R_{ex} increases, more transmission opportunities are granted for each packet and consequently more channel-caused packet losses can be reduced, which in turn results in less video freezes at the video receiver. The use of FEC can also significantly reduce video freezes but at the cost of sending a lower video bit rate and generally lower quality video due to its high overhead, and FEC typically is disabled in commercial WebRTC product as explained in Section 2.4. Comparing Fig. 13 (d) with Fig. 4 (b), we see that when a higher value of R_{ex} is used, our approach performs as well as FEC in terms of reducing video freezes, but does not have the high overhead problem which is explained in Section 2.3.

4.6 Impact on competing traffic

So far, we have shown that our proposed scheme indeed improves WebRTC-based video telephony over Wi-Fi. In this section, we investigate whether our scheme would affect the performance of competing traffic that does not use our approach.

Fig. 14 shows the aggregate throughput of the 8 competing stations named CMP_STA in Fig. 10. Since WebRTC behaves differently when FEC is turned off or on, we evaluate our scheme in both cases and present the results in Fig. 14 (a) and (b), respectively. As shown in Fig. 14, the aggregate throughput of the 8 competing stations does not change appreciably when our approach is used or not, even with different values of the parameter R_{ex}. This result agrees with our expectation. Changing the retry limit does not change the success probability for each single contention so that the fairness in contention among different stations is maintained. When the network is not congested, more retransmissions lead to the use of otherwise wasted network resources, and the cross traffic is not being hurt because network capacity is enough to accommodate all traffic. When the network is congested, our scheme can detect congestion and does not increase the default retry limit in order to not conceal the packet losses so that the high-layer congestion control algorithm could properly reduce the target sending rate to relieve the network congestion.

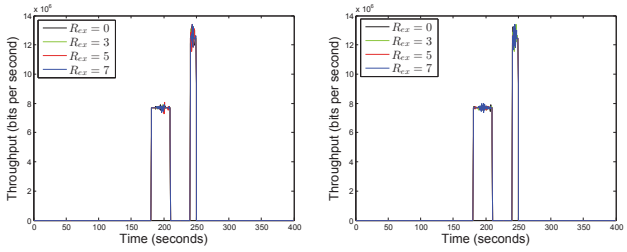


Figure 14: Aggregate throughput of competing traffic, when FEC in WebRTC is: (a) turned off; (b) turned on.

5. CONCLUSIONS

In this paper, we present a congestion-aware MAC layer adaptation algorithm to improve video teleconferencing over Wi-Fi. Inspired by the fact that the bottleneck of an end-to-end connection usually happens on the 'last-mile' wireless link, we investigate possible negative impacts when packet losses occur on the video sender's local Wi-Fi link. We develop a WebRTC-based video teleconferencing testbench which allows us to do more realistic investigations than simulation based evaluation. We confirm that packet losses on an edge Wi-Fi link may cause serious degradation to the receiver's quality of experience. Hence we propose a MAC-layer adaptation algorithm to reduce the channel-caused packet losses. We also propose a lightweight and passive congestion detection algorithm to distinguish channel-caused and congestion-caused packet losses. As most current applications (such as TCP or RTCP based applications) rely on the packet loss rate to do congestion control, our proposed MAC layer adaptation algorithm helps a higher-layer congestion control algorithm avoid unnecessary reduction in the data sending rate, but leaves congestion-caused losses intact. Experimental results confirm that our approach significantly and accurately reduces channel-caused losses so as to improve the video quality, and does not adversely impact the performance of competing traffic.

6. ACKNOWLEDGMENTS

The authors would like to thank Dharm Veer of Aricent for helping set up the testbench and collecting the data. Authors would also like to thank Dr. Ariela Zeira, Gregory Sternberg, Dr. Robert A. DiFazio and Chris Wallace of InterDigital Labs for insightful discussions and comments.

7. REFERENCES

[1] C. Lochert, B. Scheuermann, and M. Mauve. A survey on congestion control for mobile ad hoc networks: Research articles. Wireless Communications and Mobile Computing, vol.7, pp. 655 - 676, June 2007.

[2] O. Gurewitz, V. Mancuso, J. Shi, and E. Knightly. Measurement and modeling of the origins of starvation

of congestion-controlled flows in wireless mesh networks. IEEE/ACM Transactions on Networking, vol. 17, no. 6, pp. 1832 - 1845, December 2009.

[3] K. Tan, F. Jiang, Q. Zhang, and X. Shen. Congestion control in multihop wireless networks. IEEE Transactions on Vehicular Technology, vol. 56, no. 2, pp. 863 - 873, March 2007.

[4] S. Rangwala, A. Jindal, K.-Y. Jang, K. Psounis, and R. Govindan. Neighborhood-centric congestion control for multihop wireless mesh networks. IEEE Transactions on Networking, vol. 19, no. 6, 2011.

[5] A. Al Islam and V. Raghunathan. End-to-end congestion control in wireless mesh networks using a neural network. in IEEE WCNC, 2011.

[6] S. Prasanthi and S.-H. Chung. An efficient algorithm for the performance of tcp over multi-hop wireless mesh networks. in ITNG, 2010.

[7] K. Y. Lee, K.-S. Cho, and B.-S. Lee. Cross-layered hop-by-hop congestion control for multihop wireless networks. in MASS, 2006.

[8] X. Wang and D. Perkins. Cross-layer hop-by-hop congestion control in mobile ad hoc networks. in IEEE WCNC, 2008.

[9] H. Kim, S. Yun, H. Lee, etc. WLC34-1 A Simple Congestion-Resilient Link Adaptation Algorithm for IEEE 802.11 WLANs. Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE

[10] H. J. Xiao, Y. P. Wei, etc. Congestion-adaptive and queue reservation scheme for delay-constrained video streaming over IEEE 802.11 networks. 5th International Conference on Visual Information Engineering (VIE), 2008.

[11] H. Zhou, D. Hoang, P. Nhan, etc. Introducing feedback congestion control to a network with IEEE 802.11 wireless LAN. Wireless Telecommunications Symposium, 2004.

[12] P.A.K. Acharya, A. Sharma, etc. Congestion-Aware Rate Adaptation in Wireless Networks: A Measurement-Driven Approach. Proc.IEEE SECON, pp.1 - 9, 2008.

[13] L. Bononi, M. Conti, etc. Runtime Optimization of IEEE 802.11 Wireless LANs Performance. IEEE Transactions on Parallel and Distributed Systems, Vol. 15, Issue 1, Jan. 2004.

[14] An-Chih Li, Ting-Yu Lin, Ching-Yi Tsai. ARC: Joint Adaptation of Link Rate and Contention Window for IEEE 802.11 Multi-rate Wireless Networks. SECON 2009.

[15] Bianchi G. Performance analysis of the IEEE 802.11 distributed coordination function. IEEE Journal on Selected Areas in Communications 2000; 18(3): 535 - 547.

[16] M. H. Lu, P. Steenkiste and T. Chen. A time-based adaptive retry stategy for video streaming in 802.11 WLANS. Wireless Communications and Mobile Computing, 2007.

[17] S. Floyd, M. Handley, J. Padhye, etc. TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 5348, September 2008.

[18] A. Jardosh, K. Ramachandran, K. Almeroth, and E. Belding-Royer. Understanding Congestion in IEEE 802.11b Wireless Networks. In Proc. of IMC, Berkeley, CA, Oct 2005.

[19] Y. Hu and D. Johnson. Exploiting Congestion Information in Network and Higher Layer Protocols in Multihop Wireless Ad Hoc Networks. In Proc. of ICDCS, Tokyo, Japan, Mar 2004.

[20] WebRTC, "http://www.webrtc.org".

[21] OPNET, SITL, https://support.riverbed.com/bin/support/static//doc/opnet/17.5.A/online/modeler_17.5_PL5/Tutorials/wwhelp/wwhimpl/common/html/wwhelp.htm#href=sitl_tut_1.html&single=true

[22] L. D. Cicco, etc. Understanding the Dynamic Behaviour of the Google Congestion Control for RTCWeb. 20th International Packet Video Workshop (PV), 12-13 Dec. 2013.

[23] S. Holmer, M. Shemer, and M. Paniconi. Handling packet loss in WebRTC. IEEE ICIP, 2013

[24] H. Schulzrinne, S. Sivaranab, etc. RTP: A Tranport Protocol for Real-Time Applications. RFC3550, Standard, 2003

[25] S. Floyd, M. Handley, etc. TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 5348, 2008

[26] H. Bobarshad, M. van der Schaar, etc. Analytical Modeling for Delay-Sensitive Video Over WLAN. IEEE transactions on Multimedia, VOL. 14, NO. 2, APRIL 2012

[27] C. M. Chen, C. W. Lin, Y. C. Chen. Cross-Layer Packet Retry Limit Adaptation for Video Transport over Wireless LANs. IEEE transactions on circuits and systems for video technology, VOL. 20, NO. 11, November 2010.

[28] S. Lohier, Y. G. Doudane, G. Pujolle. MAC-layer Adaptation to Improve TCP Flow Performance in 802.11 Wireless Networks. Proc. IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), Montreal, Canada, 19-21 June 2006.

[29] C. M. Chen, C. W. Lin, Y. C. Chen. Packet Scheduling for Video Streaming over Wireless with Content-Aware Packet Retry Limit. In IEEE 8th Workshop on Multimedia Signal Processing, Oct 2006.

[30] N. C. Tas, T. Nadeem, A. K. Agrawala. Making Wireless Networks MORAL. INFOCOM, 2011 Proceedings IEEE

[31] Y.-C. Chen, E. M. Nahum, R. J. Gibbens, and D. Towsley. Measuring cellular networks: Characterizing 3g, 4g, and path diversity. Technical report, UMass Amherst Technical Report: UM-CS-2012-022.

[32] H. Lundin, S. Holmer, etc. A Google Congestion Control Algorithm for Real-Time Communication, draft-alvestrand-rmcat-congestion-02, 2014. http://tools.ietf.org/html/draft-alvestrand-rmcat-congestion-02.

[33] D. Bertsekas and R. G. Gallager. Data Networks(2nd edition). Prentice Hall, 1992, ISBN 0132009161.

[34] M. Franceschetti and R. Meester. Random Networks for Communication. Cambridge University Press, 2008.