Informed Bandwidth Adaptation in Wi-Fi Networks using Ping-Pair

Rajdeep Das[†], Nimantha Baranasuriya[*], Venkata N. Padmanabhan[†], Christoffer Rödbro[‡], Seth Gilbert[*]
[†]Microsoft Research India, [*]National University of Singapore, [‡]Microsoft Skype

ABSTRACT

Bandwidth adaptation for real-time streaming applications is typically designed to be conservative, since pushing for higher bandwidth could be counterproductive if it means an increased latency. However, such bandwidth adaptation operates based on the "symptoms" of congestion (e.g., increased delay) without knowing the underlying cause (self-congestion vs. cross-traffic). In this paper, we consider this problem in the context of Wi-Fi networks and introduce a novel technique, `Ping-Pair`, to measure and attribute congestion. We have integrated `Ping-Pair` into the popular Skype audio-video conferencing application to enable improved bandwidth adaptation dubbed `Kwikr`, using which we have conducted controlled experiments and also randomized A/B tests in a production setting.

ACM Reference Format:
Rajdeep Das[†], Nimantha Baranasuriya[*], Venkata N. Padmanabhan[†], Christoffer Rödbro[‡], Seth Gilbert[*]. 2017. Informed Bandwidth Adaptation in Wi-Fi Networks using Ping-Pair. In *CoNEXT '17: CoNEXT '17: The 13th International Conference on emerging Networking EXperiments and Technologies, December 12–15, 2017, Incheon, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3143361.3143390

1 INTRODUCTION

Real-time interactive streaming has been growing in importance, spanning both traditional applications such as audio-video (AV) conferencing (e.g., Skype, Google Hangouts) and newer ones such as cloud-based app streaming (e.g., Amazon AppStream) and gaming (e.g., Sony PlayStation Now). Such applications are highly sensitive to network delay, jitter, and packet loss.

Fluctuation in the available network bandwidth is particularly challenging. Unlike on-demand streaming, where a multi-second playout buffer can be used to absorb and smooth out much of this variation, the tight deadline for real-time interactive streaming (e.g., an RTT under 300 ms for VoIP and 60-100 ms for gaming) rules out a large playout buffer. Therefore, it becomes critical to estimate network bandwidth and track its variation effectively.

Sophisticated bandwidth estimation schemes have been developed for real-time streaming applications. Unlike TCP, which primarily cares about throughput, bandwidth estimation for real-time streaming cares critically about latency too (see Section 2).

The challenge, however, is that bandwidth estimation for real-time streaming, like TCP congestion control, operates end-to-end, without direct knowledge of the internal state of the network. This can lead to highly suboptimal behaviour. For instance, when congestion is due to cross-traffic, not self-congestion, the overly conservative strategy of decreasing the transmission rate sharply (more so than TCP would) would likely hurt the real-time flow without yielding any benefit in terms of reduced latency (see Figure 1).

In this paper, we present a new probing mechanism, `Ping-Pair`, and an improved bandwidth adaptation scheme, `Kwikr`, designed to address this challenge, specifically in the increasingly common setting of Wi-Fi-connected clients. The Wi-Fi link is often, though not always, the bottleneck link that determines the fate of the end-to-end real-time flow [31]. Moreover, since the client is directly connected to this link, it is in a position to make local observations, which enables faster and more effective end-to-end bandwidth adaptation.

In particular, when there is an increase in end-to-end delay, we seek to determine whether the increase is due to congestion at the Wi-Fi downlink and if so whether the congestion is self-induced or due to cross-traffic. Note that cross-traffic could either be in the same Wi-Fi network or in other, neighbouring co-channel networks causing contention, and we wish to address both cases. To this end, we have developed a novel technique called `Ping-Pair` [1] that enables a client to determine the amount of delay on the Wi-Fi AP's downlink. `Ping-Pair` takes advantage of the 802.11e or Wireless Multimedia Extensions (WMM) support in APs to send a pair of pings (which, as we report in Section 5.5, is quite widespread), one at high priority and the other at normal priority. The time difference in the arrival of the two ping responses yields an estimate of the Wi-Fi delay — due to queuing at the AP and due to contention with co-channel cross-traffic. Furthermore, by counting the packets of the flow of interest (e.g., `Skype`) sandwiched between the two ping responses, we determine how much of the delay is self-induced versus due to cross-traffic. Armed with this information, our improved bandwidth adaptation scheme, `Kwikr`, modifies bandwidth estimation to avoid backing off in an overly conservative manner (i.e., more conservatively than TCP) when the congestion is due to cross-traffic; instead, the backoff in such a case is performed in line

Das is now with UC San Diego. Baranasuriya was an intern at Microsoft Research India and is now with Ackcio. Rödbro is now with Google.

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 the author(s) 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.
CoNEXT '17, December 12–15, 2017, Incheon, Republic of Korea
© 2017 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-5422-6/17/12…$15.00
https://doi.org/10.1145/3143361.3143390

[1] Distinct from and not be be confused with the "packet-pair" technique from the literature, as discussed in Section 5.2.

with TCP, which yields performance gains relative to the overly conservative approach, while ensuring safe behaviour.

We have implemented Kwikr in Skype, a popular AV conferencing application with hundreds of millions of users. Our implementation, which is presently on stock Android version 7.23.0.245 onwards and includes 1500 lines of code, spans multiple modules in Skype, some platform-specific (e.g., invoking OS APIs to query the Wi-Fi link) and others platform independent (e.g., bandwidth adaptation). Separately, we have also implemented Kwikr as a standalone module on Microsoft Windows and Linux, which offers us greater control (e.g., access to raw sockets) than is available on (unrooted) Android.

We have evaluated Ping-Pair probing and Kwikr adaptation in a range of Wi-Fi settings, including offices, coffee shops, airports, conferences, etc. We have also conducted controlled experiments in the lab. Our finding is that Ping-Pair accurately estimates the congestion state of the Wi-Fi network and furthermore it enables Kwikr to improve streaming performance *safely*, i.e., without causing significant negative side-effects, either to the flow itself or to other traffic. For instance, in our controlled experiments, where we introduced congestion during a call, Kwikr achieves 20% higher throughput, while maintaining the same loss rate and round-trip time.

Kwikr, with Ping-Pair probing built-in, is part of the production Skype client for Android version 7.23.0.245 onwards and has been A/B tested with actual users. Our results from about 120,000 video calls show that there is significant downlink delay at Wi-Fi APs in the wild, much of it due to cross-traffic. Furthermore, Kwikr results in a 8.5% gain in the median bandwidth of a call, without causing an increase in either the round-trip time or the packet loss rate. Thus, Kwikr provides benefit (improved bandwidth) while being safe (no negative impact on the network).

To summarize, our main contributions here are:

- The design of the Ping-Pair technique to estimate delay at the Wi-Fi AP's downlink and attribute the same.
- The design of Kwikr, which leverages Ping-Pair to enabled informed bandwidth adaptation.
- The implementation of Ping-Pair and Kwikr in the popular Skype application and results from A/B testing of these in the wild.

2 RELATED WORK

We briefly survey several strands of related work.

TCP congestion control: TCP congestion control [25] is based on an additive-increase multiplicative-decrease (AIMD) policy [12]. With its focus on throughput, TCP tends to fill network buffers, leading to increased delay. Variants of TCP react to delay as well [9, 38], though reducing delay has not been the primary impetus.

TCP congestion control operates end-to-end (as does, say, PCP [2]). There has also been work, starting with the pioneering DECbit effort [35], where routers mark packets to provide explicit congestion notifications (ECN) [17, 18, 30]. Outside of data centers [1], ECN has not seen much deployment. In Kwikr, we go beyond the end-to-end view by having clients probe the local Wi-Fi link, rather than depending on explicit signals from the network.

Real-time streaming: Real-time streaming applications often have a tight requirement on latency, and so typically operate over UDP rather than TCP, with an application-layer congestion control scheme such as TCP-Friendly Rate Control (TFRC) [21]. However, TFRC, by itself, does not address the problem of delay, since queues can still build up.

Several real-time applications have gone further in focusing on low latency. The general approach is to measure delay and inter-packet spacing, an increase in which is presumed to be a sign of queues building up, causing the sender to back off. These applications tend to respond conservatively to congestion, compared to TCP; see experimental verification for Skype [43] and Google Congestion Control [13] (one of the schemes under consideration for WebRTC [8]).

In Kwikr, the Ping-Pair technique exposes the queuing behavior at the Wi-Fi link so we can avoid an unnecessarily conservative reaction when it is not warranted.

Bandwidth estimation: There is a large body of work on bandwidth estimation [15, 24], including in wireless networks [7, 33]. However, these techniques typically involve sending a specially-crafted stream of measurement packets (e.g., a packet train), which could impose significant overhead. Hence, TCP and real-time streaming protocols rely instead on estimating bandwidth based on an application's data stream itself.

There has also been work on estimating link quality [3] and link capacity [16] based on error rates. Sprout [42] and Performance-Oriented Congestion Control (PCC) [14] use stochastic forecasting and randomized controlled trials, respectively, for bandwidth estimation and congestion control in wireless networks. The context for these is different from Kwikr; e.g., Sprout focuses on cellular networks with per-station queues (not Wi-Fi networks) and treats the network as a block box (instead of peeking at the Wi-Fi link to derive hints). With regards to hints, [36] also taps sensors such as GPS and accelerometer for mobility-triggered MAC-layer rate adaptation and AP selection, whereas Kwikr focuses on the impact of delays introduced by congestion and contention on the Wi-Fi downlink.

Wireless setting: There has been work on studying other wireless-related issues such as the impact of random wireless errors on TCP congestion control [4–6, 11, 19], the impact of wireless handoffs on TCP performance [10, 20] and AP mechanisms to ensure low latency [22]. Our focus on addressing Wi-Fi congestion and doing so without AP modifications is orthogonal to this prior work.

Network diagnosis: There has also been work on network diagnosis, including on isolating Wi-Fi problems. For example, [39] considers the problem of disambiguating between problems on the Wi-Fi link, the access link, and the WAN path, while [28] employs user-level active probing to detect Wi-Fi pathologies such as congestion and low SNR. However, both these approaches involve using the Wi-Fi AP or a directly connected (wired) node as a vantage point, which is a practical hindrance that we avoid in Kwikr. While WiSlow [32] also avoids the requirement of such a vantage point, it depends on potentially disruptive steps such as putting the client's Wi-Fi NIC in monitor mode. In contrast, Kwikr works with just standard OS APIs.

3 MOTIVATION

In this section, we show that existing real-time streaming applications do not respond well to congestion. We consider three popular AV conferencing applications — Skype, FaceTime, and Hangouts. For Skype, we have access to an instrumented client that provides detailed logs of metrics pertaining to a call, including, throughput and per-packet RTT. For FaceTime and Hangouts, we measure throughput via Wireshark [40] packet captures.

Consider Figure 1(a), which shows the data rate of both a Skype AV stream and of a foreground TCP flow. Congestion, in the form of cross-traffic (generated by 6 devices, each performing a TCP bulk transfer), is introduced and withdrawn on the Wi-Fi bottleneck link at the points in time marked by the shaded area. At the onset of congestion, the data rate of both Skype and TCP plummets, but TCP soon recovers to about 1.5 Mbps while Skype remains stuck at the much lower level of about 200 Kbps.

Skype adopts a conservative approach, presumably to limit queuing delay and avoid packet loss. However, such an approach does not help, with the RTT remaining high throughout the congestion episode (Figure 1(d)) despite the sharp cutback in Skype's data rate. The reason is that the congestion is due to other traffic, not self-congestion due to the Skype flow itself. So Skype is being needlessly conservative, while deriving no benefit.

Similarly, even when the congestion episode has concluded, Skype is slow to recover, as shown in Figure 1(a), because it is unaware of the rapid change in the congestion state of the wireless link.

The other two AV conferencing applications—FaceTime and Hangouts—also show similar conservative behavior, as depicted in Figures 1(b) and 1(c), taking 10s of seconds to recover after congestion ends. [2]

These experiments highlight a key problem in the state-of-the-art bandwidth adaptation techniques used by real-time streaming applications like Skype, FaceTime and Hangouts. These techniques operate end-to-end, and so are oblivious to the cause of congestion, leading to sub-optimal behaviour.

4 OVERVIEW OF KWIKR

We begin with an overview of Kwikr. Figure 2 shows the high-level architecture. Kwikr sits in between the Wi-Fi network interface and applications. It comprises a set of detectors that observe the conditions over the Wi-Fi link and turn these into actionable hints for applications. While we focus on just congestion in this paper, the Wi-Fi-specific hints also pertain to link quality fluctuation, handoffs, etc.

To demonstrate the utility of the congestion-related hints, in particular, we focus on Skype, where we modify its bandwidth adaptation strategy accordingly. Note that the bandwidth estimation happens at the receiver, which then conveys its estimate to the sender. So our interest is in the downlink direction on Wi-Fi.

The Wi-Fi last-hop is often the bottleneck link in the end-to-end path. Therefore, the goal of Kwikr is to take advantage of the direct attachment of the client to the Wi-Fi link to expose the congestion and its cause, e.g., the contribution of the real-time application to it. To avoid the practical difficulties of sniffing on the Wi-Fi channel, Kwikr focuses on detecting the delay due to queuing and contention at the downlink of the Wi-Fi AP as a sign of congestion. To this end, we present a novel Ping-Pair technique, which takes advantage of Wireless Multimedia Extensions (WMM) support in Wi-Fi APs to estimate the delay at the AP's downlink and the contribution of the flow of interest.

Based on Ping-Pair, we estimate the contribution of the flow of interest (e.g., Skype) to the delay at the Wi-Fi link and use this information to modulate the bandwidth estimation of the flow. In Skype, we do this by modulating the response of the Kalman filter used for bandwidth estimation. Specifically, if there is an increase in the delay, but Skype's contribution to it is small, the increase in delay is attributed to cross-traffic, whether in the same cell or in other co-channel cells, and treated as noise rather than a sign of self-induced congestion. Note that Skype would still be responsive to such indications of congestion like any network flow should but would avoid the overly-conservative response discussed in Section 3, since such a response would do little to reduce delay given that the congestion is due to other traffic.

Note that although our implementation focuses purely on Skype, such modifications can be applied in any real-time streaming application to improve its performance by allowing it to be "Wi-Fi-aware" using the techniques we introduce in this paper.

5 DETECTING WI-FI CONGESTION

A congested Wi-Fi network is one where the channel is constantly busy because one or more nodes always has packets to send. These transmissions could either be by nodes in the same network or by nodes in other, co-channel networks causing interference. We devise Ping-Pair, which enables a Wi-Fi-connected client to measure and attribute the delay on the Wi-Fi downlink at the AP, without requiring any AP modification or disruptive steps such as monitor mode for sniffing at the client. Ping-Pair leverages Wi-Fi packet prioritization, which we provide background on next.

5.1 Packet Prioritization

The IEEE 802.11e standard proposes a set of QoS enhancements to IEEE 802.11 through the Enhanced Distributed Channel Access (EDCA) mechanism. EDCA supports four traffic classes (also called as Access Categories): Background, Best Effort, Video, and Voice, in increasing order of priority.

The Wi-Fi alliance's Wi-Fi Multimedia (WMM) specification contains a subset of the enhancements proposed by 802.11e. WMM requires an AP to support EDCA and to maintain four queues in the downlink direction—one for each access category. The AP always schedules frames in a higher priority queue before those in a lower priority one. Furthermore, EDCA adjusts the AIFS (Arbitration Inter-Frame Spacing) and CWmin and CWmax (minimum and maximum contention windows) parameters to ensure that high-priority traffic (e.g., traffic marked as voice) experiences minimal contention delay.

The DiffServ Code Point (DSCP) field in the IP header [29] (formerly the Type of Service (TOS) bits) is mapped to the WMM priority classes at a Wi-Fi AP. However, these bits are typically honoured only within the administrative boundaries of networks and are often reset when a packet crosses an inter-AS boundary.

[2] We do not have access to detailed metrics for FaceTime or Hangouts, and hence cannot plot RTT.

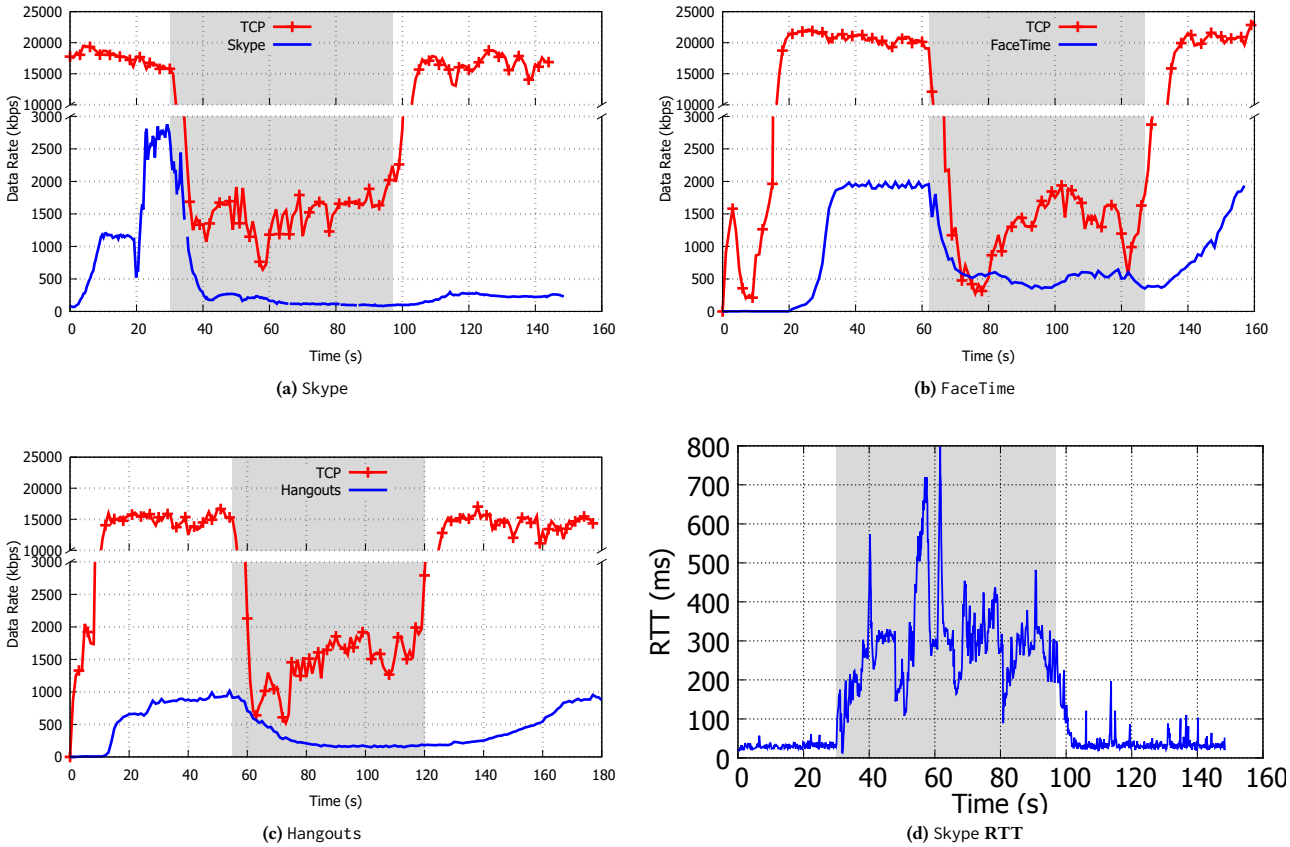


Figure 1: Congestion response of (a) Skype, (b) FaceTime, and (c) Hangouts vis-a-vis TCP. (d) depicts the RTT of Skype packets. The shaded region in each plot depicts the period when Wi-Fi congestion was introduced in the form of TCP bulk transfers.

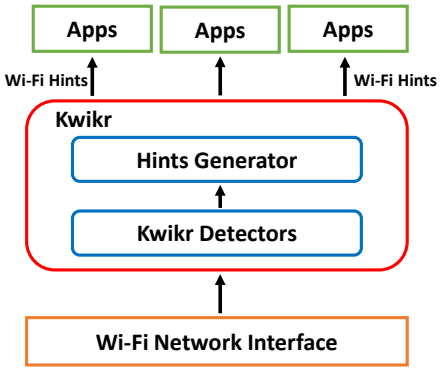


Figure 2: A high-level architectural of Kwikr.

So virtually all of the Internet traffic arriving at a Wi-Fi AP has the default, best-effort priority setting. We leverage this in our design of Ping-Pair.

5.2 Ping-Pair

To estimate the delay on the Wi-Fi downlink, the client sends a pair of back-to-back ping requests (ICMP Echo Request) destined to the AP, the one with the TOS set to high priority (0xb8) and the other one with normal priority (0x00). These priority values correspond to the Best Effort and Voice traffic classes of 802.11e, respectively. The corresponding ping responses (ICMP Echo Reply) also have the TOS bits set to high priority and normal priority, respectively, as required in the ICMP standard [34]. Therefore, the high-priority ping response ends up at the head of the AP's downlink queue (since, as noted above, TOS remains set at the default, best-effort level for virtually all traffic arriving at the AP over the Internet), and moreover EDCA ensures that this packet experiences minimal contention delay due to other traffic, whether in the same network or other, co-channel networks.

On the other hand, the normal-priority ping packet gets enqueued at the tail, as is the case usually, and experiences the same delay due to queuing and contention that any normal packet would. This is illustrated in Figure 3(a). (Note that the AP would typically implement multiple prioritized queues; however, the details of the AP's implementation are not relevant here, so we imagine the

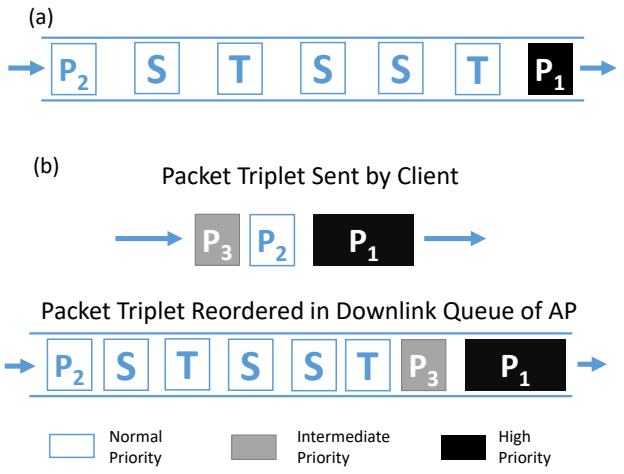


Figure 3: The downlink queue at the AP in various cases: (a) for `Ping-Pair`-based queue estimation, and (b) for detection of WMM prioritization. Note: P = ping requests/responses, with the subscripts indicating the sequence of transmission by the client; S = Skype packets; T = TCP or other cross-traffic.

conceptually-simple model of a single queue from which packets are served in priority order.)

When the ping responses are received at the client, the difference in their arrival times corresponds to the delay, T_q, at the Wi-Fi downlink. In a setting where we are unable to measure the actual arrival times (see Section 7), we approximate it as the difference in the ping times. By applying a threshold on this queuing delay estimate (see below for implementation details), we determine whether there is congestion.

Note that to ensure a valid `Ping-Pair` measurement, the two ping responses must be enqueued at the AP's downlink concurrently. So the normal-priority ping is sent first, followed immediately by the high-priority one, and `Ping-Pair` measurements are only made in cases where the high-priority response arrives before the normal-priority one. In our experiments, we find that such valid measurements occur 98% of the time in cases where the Wi-Fi downlink is congested.

Despite the similarity in names, our `Ping-Pair` technique is novel and distinct from the "packet-pair" technique in the literature in several ways: (a) the construction of `Ping-Pair` requires different priority settings for the two ping packets unlike with packet-pair, (b) `Ping-Pair` is used to estimate downlink delay whereas packet-pair is used to estimate bandwidth, and (c) `Ping-Pair` can also help attribute queuing delay to self-congestion vs. cross-traffic, which we discuss next.

5.3 Self-Congestion vs. Cross-Traffic

To attribute congestion, i.e., identify its cause, we want to estimate how much of the delay is due to packets from the particular flow of interest. We do this in two steps. First, the client determines the queue *occupancy* due to the flow of interest by counting the number of packets, n_a, from that flow received in between (i.e., sandwiched between) the high-priority ping response and the normal-priority ping response. For instance, if the client is interested in estimating the queue occupancy due to the incoming packets of a Skype flow, it would just have to count the Skype packets that are sandwiched between the ping responses, for instance, $n_a = 3$ packets in Figure 3(a).

Second, to estimate the contribution, T_a, of the Skype packets to the *delay*, we add up the transmission time of each packet (calculated by dividing the packet's size, s_a, by the MAC layer data rate, R, at which the packet was transmitted over the channel, both of which are readily available) and the channel access delay, t, incurred by the packet (the estimation of which is discussed below). So $T_a = n_a(\frac{s_a}{R} + t)$ In turn, this allows us to estimate the delay, T_c, due to cross-traffic as $T_c = T_q - T_a$. Thus, we can not only estimate the total Wi-Fi downlink delay but also attribute it to self-congestion vs. cross-traffic.

5.4 Estimating Channel Access Delay

To estimate the channel access delay, the client sends a pair of pings to the AP, but both at normal priority. This ensures that the two responses are also marked as normal priority (just like other traffic for which we wish to estimate the channel access delay) and also maximizes the chance that the responses are queued up one right behind the other at the AP's downlink. If the two ping responses are, in fact, queued up one right behind the other (as we check in the next paragraph below), the time gap between their reception at the client minus the transmission time of the second ping response would yield an estimate of the channel access time.

To filter out spurious measurements, the client only considers instances where the two ping responses have consecutive 802.11 frame sequence numbers, which ensures that these packets were transmitted by the AP one right after the other, with no intervening packets in the AP's local queue. (Of course, there could be intervening transmissions from other contending nodes, which is indeed what would contribute to the channel access delay.) Also, if either ping response has the 802.11 retransmit bit set, we disregard the measurement. However, given the challenges of accessing low-level information such as 802.11 frame sequence numbers and retransmit bit, we use a fixed value of channel access delay in our implementation, as noted in Section 7.

5.5 Checking for Prioritization

We rely on WMM prioritization to be enabled on the wireless AP. A client can check this by sending a triplet of back-to-back ping packets, as depicted in Figure 3(b): a large, high-priority ping followed by two small pings, the first a normal-priority one and the second an intermediate-priority one. If WMM prioritization is enabled, the order of responses to the small pings would tend to get reversed. On the other hand, if WMM prioritization is not enabled, the order of responses would tend to be the same as the order of requests.

We tested this technique in six different Wi-Fi networks with APs from various manufacturers. Checking for reversal in at least 3 of 5 runs led to accurate detection.

We also conducted a measurement study where we recruited users on Amazon Mechanical Turk to run a tool containing this test. [3] We obtained data from 171 unique Wi-Fi access points from 166 users across 14 countries. We found that WMM prioritization was enabled on 77% of these APs, which indicates that Ping-Pair has wide applicability.

5.6 Robustness of the Ping-Pair Technique

A potential concern with the Ping-Pair technique is that one or more packets in the ping-pair may be delayed due to a failed transmission (e.g., due wireless errors), leading to a later retransmission. This could cause the Ping-Pair estimate to be artificially inflated.

A conceptually straightforward way to avoid such inflation would be to disregard any measurements where the 802.11 retransmit flag is set on either or both of the received ping responses. However, since a user-level application is unlikely to have access to this low-level information, we introduce an alternative dual-Ping-Pair technique.

In Dual-Ping-Pairs, two sets of ping-pair packets are sent simultaneously. If there is a gap between the two high-priority ping responses or a gap between the two normal-priority packets, then we can detect a retransmission problem and discard the entire measurement.

The remaining problematic case is when the first of the two normal-priority packet is delayed due to retransmissions, but the second is not. Here there are two possibilities: the packet loss may be caused by some sustained event (e.g., a long period of interference) that impacts many packets, or the packet losses are uncorrelated and due to random noise. If there is some sustained event, then it is likely that all the packets will be affected, and the problem will be detected by the gaps described above. On the other hand, if the packet loss is uncorrelated and random, then it is unlikely to occur repeatedly and hence it will have little impact on the congestion control mechanism. Since the congestion control mechanism only responds to a sustained signal, and since the probability of long retransmission delays decreases geometrically, the occasional random misdetection will not have a significant impact. We can see this both in practice and analytically.

To validate the effectiveness of the dual-Ping-Pair technique, we ran an experiment where the client started close to the AP (strong link), quickly moved away and stayed away for a while (weak link), before moving back towards the AP (strong link). We discard any dual-Ping-Pair measurement where the estimates from the two pairs differs by more than 5 ms. In Figure 4, we report the maximum number of link-level transmissions (i.e., # retransmissions + 1) undergone by any of the dual-Ping-Pair request or response packets each second, and also the maximum value of the ping-pair queuing delay estimate each second as well as an EWMA-smoothed value of the queuing delay estimate (which is used in Skype with Kwikr). We see that though the maximum number of link-layer transmissions of a packet is as high as 6 (signifying a poor link), the dual-Ping-Pair technique is able to filter out extraneous delay spikes. The filtered ping-pair estimates are all smaller than 5 ms, and the smoothed value is even smaller. The absence of filtering, on the other hand, resulted in spikes above 100ms in our experiments.

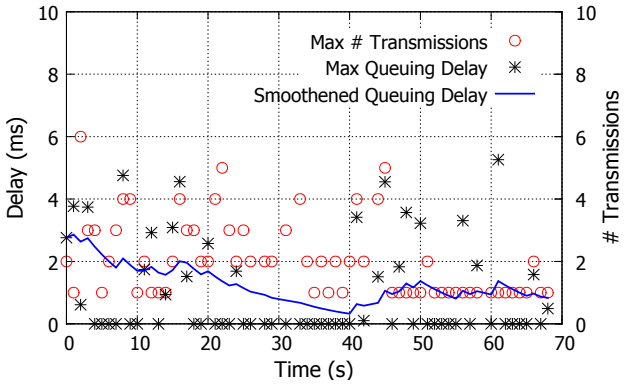


Figure 4: Impact of link-layer retransmissions on queuing delay estimates using the dual-Ping-Pair technique.

Simplified Analysis: We can also analytically examine the probability that retransmissions have a significant impact. The following provides some intuition for such analysis. Consider, say, the case where the probability of a packet loss (and hence retransmission) is independent with probability p. (The correlated case is easier, as it will be detected and filtered by the dual-Ping-Pair.)

The dual-Ping-Pair filtering fails if either the first low-priority packet is itself delayed, or if the intervening packets (whether Skype or cross-traffic) are delayed by a significant factor. Since the probability of a packet loss is p, we can readily bound the probability that the first low-priority packet is retransmitted even once by p.

To calculate the probability of the intervening packets being delayed, assume there are n intervening packets. Assume that packets are retransmitted using some form of exponential backoff scheme, e.g., if a packet is retransmitted k times, then it takes time approximately 2^k times longer than if it had not been retransmitted at all. We can estimate that if the intervening packets are delayed in a way that triples the length of the gap, then this delay may lead to a misattribution. Since the packet retransmissions rely on exponential backoff, it will take *at least* $\log(2n)$ retransmissions (in the worst-case) to increase the gap by $2n$ (i.e., yielding a gap of size at least $3n$). Thus the probability of the intervening packets increasing sufficiently is at most $np^{\log(2n)} \leq 2p$. (The multiplicative factor of n arises from the possibility that any one of the n packets may be delayed this many times, which is the worst-case.)

Thus the probability of a given dual-Ping-Pair failing is at most $p + 2p = 3p$ (by a union bound). We observe that the congestion control mechanism only responds to sustained signals, e.g., if 3 consecutive dual-Ping-Pair tests fail due to retransmissions. The probability of t consecutive dual-Ping-Pair tests failing is at most $(3p)^t$. Assuming a (high) transmission failure rate of 10%, the probability of three consecutive bad tests is less than 3%. Thus, it is unlikely for the congestion control mechanism to be impacted by sustained Ping-Pair errors.

[3] We obtained IRB approval for this study.

5.7 Ping-Pair in Alternative Wi-Fi Setups

We evaluated the Ping-Pair technique in enterprise settings, including an office and a university, where the Wi-Fi service uses thin APs managed by central controllers. The Wi-Fi deployments at the office and the university are based on gear from Aruba and Cisco, respectively. In such thin AP settings, the default gateway IP that the client pings does not belong to the AP as discussed above, but to a middlebox in the network managed by the controller. Despite this, in both settings, Ping-Pair provided accurate detections because the TOS bits of the IP header were preserved in the network. This suggests that Ping-Pair is effective in thin AP settings.

Wi-Fi range extenders are sometimes used in homes to extend Wi-Fi coverage. We evaluated our Ping-Pair technique in such an environment too. We used two TP-Link APs for this experiment; one as a regular AP and the other as a range extender. We connected a client to the range extender and conducted Ping-Pair trials. As the range extender setup did not change the TOS bits of the IP header and also supported WMM prioritazation, our Ping-Pair technique provided accurate detections in this setting as well.

6 ADAPTATION BASED ON PING-PAIR

In this section, we outline how Ping-Pair measurements can be leveraged for bandwidth control. As the Ping-Pair measurements relate only to the AP downlink, they alone do not form a basis for end-to-end bandwidth control. However, as we shall see, they provide useful guidance in improving existing end-to-end bandwidth control mechanisms, such as the ones used by real-time streaming applications like Skype. Conceptually, using Ping-Pair we can modify the bandwidth control mechanism to take into account whether delays at the Wi-Fi downlink is caused by the application that the mechanism governs or not. If delays are due to other traffic in the Wi-Fi network, being overly conservative in reducing transmission load is unlikely to help; it would only have a negative impact on the user experience in terms of audio and video quality but not yield gains in terms of reduced delay.

Incorporating Ping-Pair to adjust existing end-to-end bandwidth control mechanisms is highly implementation specific. Commonly, control mechanisms for real-time streaming applications are based on one-way packet delay measurements, either as a congestion signal or as input to a bandwidth estimation algorithm, see e.g. [23, 26]. For such techniques, an obvious modification would be to simply remove the cross-traffic contribution from the measured end-to-end delays, that is, $d \leftarrow d - T_c$, where T_c is the delay due to other sources (i.e., cross-traffic), as determined by the Ping-Pair technique according to Section 5.3. However, this modification is challenging in practice since the Ping-Pair measurements tend to be much less frequent (twice per second in our implementation) than the one-way delay samples (which are obtained for every packet received, typically with an inter-packet spacing of 20 ms).

For our target application, Skype, we use a more subtle approach to adjust its bandwidth control mechanism based on Ping-Pair measurements. A full description of Skype's bandwidth estimation mechanism (which is prior work that we build on) is outside the scope of this paper, but in short Skype estimates path bandwidth by Kalman filtering, modeling the end-to-end delays as a function f of the available bandwidth of the network path (see [37]):

$$d(k) = f(BW(k)) + e(k) \tag{1}$$

where k is a packet index, $BW(k)$ is the serving bandwidth, and $d(k)$ is the packet one-way delay, calculated from transmission and reception timestamps and compensated for sender/receiver clock offset and channel propagation delay through a minimum tracking mechanism. $e(k)$ is the Kalman filter observation noise.

At a high level, a Kalman Filter estimates the state of a system, combining a predictive model with ongoing noisy observations. The idea is to integrate the Ping-Pair information by treating congestion by other sources (i.e., a high T_c) as an indication that the observations $d(k)$ are noisier than normal since the cross-traffic is corrupting the measurements. In a Kalman filter framework, that can be achieved by increasing the observation noise variance parameter σ_e^2 in accordance with T_c; the higher the T_c is, the larger the noise variance. The effect is that the bandwidth estimator makes a smaller-than-usual reaction to the increased delay.

Skype uses an Unscented Kalman filtering (UKF) [27, 41] of the leaky bucket state-space model (see [37]. UKF is based on the Unscented Transform, a method for computing the statistics of a random variable that undergoes non-linear transformation; this is because the function f in Equation 1 is non-linear. It operates by sampling the state distribution at selected *sigma points* χ to obtain a Gaussian representation, which is amenable for propagation through a Kalman Filter. There are $1 + 2L$ such sigma points, one central χ_0 and two for each of the L random variables in the Kalman filter. The observation noise is one such random variable and following the notation of [41], the two corresponding sigma points are calculated according to:

$$\chi_e^{(+/-)} = \chi_0 \pm \sqrt{\alpha^2 L \sigma_e^2} \tag{2}$$

where $\alpha = 1e-3$ is a typical parameter value. This equation is our attack point: by increasing the observation noise variance parameter σ_e^2, we make the Kalman filter less sensitive to congestion. As an additional trick, we do this only for the '+' sigma point, which corresponds to modeling the observation noise as having non-zero positive mean:

$$\chi_e^{+} = \chi_0 + \sqrt{\alpha^2 L \left(\sigma_e^2 + \beta T_c^2\right)} \tag{3}$$

where β is an empirically tuned "noise scaling" factor. The larger the β is, the less the Kalman filter would respond to increased delay due to cross-traffic; we found $\beta = 4$ to be adequate. Thus, by modeling the delay observation noise as having non-zero positive mean, we signal that not all delay is caused by Skype, making the estimator converge towards a higher bandwidth. And by increasing the observation noise variance, we introduce greater uncertainty, causing less weight to be given to the delay observation, thus slowing down the bandwidth adaptation speed.

7 IMPLEMENTATION

We implemented [4] a full-fledged Ping-Pair tool on Windows and Linux, where we had access to low level APIs for creating raw

[4]For a standalone implementation of ping-pair, please visit https://www.microsoft.com/en-us/research/project/pindrop/.

sockets and obtaining precise time of send and arrival of packets. However, in order to deploy it on Skype (specifically on Skype for Android), we created a simpler implementation tailored for the restrictive Android platform.

7.1 Ping-Pair Tool on Windows

We first implemented Ping-Pair as a standalone application on Windows. Our implemention comprises $\tilde{1}200$ lines of C# code. It uses raw IP sockets to construct the necessary probes. (The User Account Control (UAC) mechanism on Windows prompts the user for the necessary privileges.) We used this standalone application for testing and also for distribution via Amazon mTurk for the study noted in Section 5.5 on checking the prevalence of WMM support in the wild (users ran the tool on their Wi-Fi-connected Windows PCs).

7.2 Ping-Pair Tool on Linux

We then implemented Ping-Pair as a standalone application on Linux, which is the OS that underlies Android. This implementation, again, uses raw IP sockets to construct the necessary ICMP packets required for the probes. Our implementation consisted of around 500 lines of C code.

We also implemented our channel access delay estimator on Linux, since it enabled us to send out back-to-back packets at a very nominal delay of a few microseconds. Our implementation, comprising $\tilde{6}00$ lines of C code, consists of two components: a probing tool and a packet capture tool. The probing tool sends out a pair of normal priority (i.e., best-effort) pings back-to-back using raw Linux sockets. The ping responses are then captured using a packet capture interface from which we infer the channel access delay. In order to accurately measure the channel access delay, we filtered out probes where either of the frames were retransmitted on the Wi-Fi link or if the IEEE 802.11 frame sequence numbers of the two ping responses were not consecutive.

7.3 Kwikr for Skype on Android

Our main implementation effort centered on the popular Skype AV conferencing application on Android. This application is largely written in C, with a UI shell in Java that loads a C library using the Java Native Interface (JNI). Our implementation spanned the following three components (the lines of code (LOC) of our implementation are noted within parentheses):

(1) **Platform Interface (PI)** (50 LOC): this is a platform-specific module responsible for interfacing with the OS to obtain the MAC data rate information and invoke the ping utility on Android.
(2) **Probing Module (PM)** (1400 LOC): this is also platform-specific and orchestrates the Ping-Pair measurements.
(3) **Bandwidth Estimator (BE)** (50 LOC): this module is platform-independent and the only component that lies in the data path, i.e., actually receives the stream packets. BE implements Kwikr's modifications to the Unscented Kalman Filter based bandwidth estimator.

The operation starts by invoking PM at the beginning of a call. PM then obtains information about the default gateway (the Wi-Fi AP), which it probes regularly to obtain estimates of the Wi-Fi downlink delay. Each probe records the start/stop time (to determine the number of packets sandwiched between the ping-pair), the Wi-Fi downlink delay estimate, and the instantaneous MAC data rate (to infer the transmission speed). This information is then passed to BE, which counts the number of Skype packets between the start and stop times, and computes the contribution of Skype to the Wi-Fi downlink delay. Finally, this information is used to adjust the UKF, as discussed in Section 6.

One practical difficulty was our inability to use raw sockets on Android without rooting the device. To get around this limitation, our implementation uses the built-in ping utility on Android, which supports the setting of the TOS bits. However, we lack precise control over the timing of the ping packets and moreover only obtain the ping times, not the interarrival time between the ping responses. Nevertheless, our estimation of the Wi-Fi downlink delay did not suffer much relative to a raw sockets based implementation. We tested a raw-socket based implementation a ping-based implementation on Android, both in an uncongested setting and found that the average ping estimates were 1 ms and 0 ms respectively, i.e. close to each other. Likewise, in a congested setting these were 180 ms and 183 ms respectively, again close to each other.

The inability to use raw sockets also meant that we were not in a position to perform the WMM detection from Section 5.5 on Android. However, this was still acceptable; lack of WMM support only means that Ping-Pair would under-estimate the Wi-Fi downlink delay and, consequently, also the contribution of cross-traffic (c in Equation 3). The net result is that Skype would fall back to its default, non-Kwikr behaviour, which is still safe, even if it is overly conservative.

The restrictive Android platform also prevented us from implementing the channel access delay estimator, as access to low level information such as IEEE 802.11 frame sequence numbers and retransmit bits were not available. Hence, we used a fixed channel access delay of 0.125 ms in our implementation, although we realize that such a fixed value is necessarily an approximation.

8 EVALUATION

We first evaluate the Ping-Pair technique, and then present results from experiments we conducted using Skype with Kwikr, which uses Ping-Pair to enable informed bandwidth adaptation.

8.1 Validating Ping-Pair

As discussed earlier, delays in a Wi-Fi downlink can occur due to congestion and contention introduced by traffic in the same network as well as by traffic in other co-channel networks through interference. We first evaluate Ping-Pair in settings where delays are caused by traffic in the same network and then discuss its effectiveness in co-channel interference settings.

Congestion. To obtain the ground truth on whether the Wi-Fi link is congested, *i.e.,* whether there is a queue build-up at the AP, we instrumented OpenWRT (Chaos Calmer 15.05) to log the number of frames in the downlink queue continuously and installed it on a Netgear WNDR3800 Wi-Fi access point. If over 90% of the log samples show a non-empty queue, we deem it to be a "persistent" queue, otherwise a "non-persistent" queue.

Table 1: Confusion matrix for congestion detection using `Ping-Pair` in 2.4 GHz band (a) and 5 GHz band (b).

(a) 2.4 GHz Band

		Kwikr Classification	
Queue Ground Truth		Non-persistent	Persistent
Non-persistent	116	107 (92.2%)	9 (7.8%)
Persistent	117	14 (12.0%)	103 (88.0%)

(b) 5 GHz Band

		Kwikr Classification	
Queue Ground Truth		Non-persistent	Persistent
Non-persistent	110	108 (98.2%)	2 (1.8%)
Persistent	140	16 (11.4%)	124 (88.6%)

We increased the number of (cross-traffic) TCP flows to other Wi-Fi clients connected to the same AP, from 0 to 7, to create varying amounts of congestion, and gathered 30 `Ping-Pair` measurements at each step. We used both 2.4 GHz and 5 GHz bands as the Netgear WNDR3800 access point supported dual-band operation. Next, we trained a 10-fold cross-validation decision tree classifier on this data and the ground truth, to obtain the thresholds of 5ms on the ping-pair delay estimate for both bands, for classification of persistent vs. non-persistent congestion.

Tables 1(a) and 1(b) show the confusion matrices of `Ping-Pair` for the 2.4 GHz and 5 GHz bands, respectively. In both bands, `Kwikr`'s congestion detection accuracy is over 90%.

This shows that the `Ping-Pair` technique, despite its simplicity, is a reliable detector of Wi-Fi congestion.

Other AP models. Quantitative classification of congestion requires the instrumentation of APs to get the ground-truth, as done above. As this is not possible on every AP, we conducted qualitative experiments on multiple APs (from LinkSys, TP-Link, Cisco, and D-Link), where we introduced congestion in the form of TCP bulk flows and tested the `Ping-Pair` classification model we constructed above. In all the settings, the `Ping-Pair` technique successfully detected the congested versus non-congested episodes.

Interference. To evaluate the effectiveness of the `Ping-Pair` technique in interference settings, we set up two APs running on the same channel. On the first AP, we connected a UDP client to the Wi-Fi link and a UDP server to the wired link. We then initiated a downlink stream of UDP packets (with an inter-packet interval of 20 ms) from the server to the client in order to simulate a Skype call and measured the one-way delay. On the Wi-Fi link of the same AP, we conducted `Ping-Pair` trials every 200 ms. We connected 6 clients running 20 parallel TCP bulk transfers to the second AP and congested it for 30 seconds during the experiment.

As Figure 5 shows, congestion in the neighbouring AP increases the one-way-delays of the UDP stream because of co-channel interference. It also shows that our `Ping-Pair` detector's delay estimates increase during the time period the AP was experiencing interference. Note that to combat clock offset, the one-way-delay shown in the plot has been normalized by subtracting out the minimum one-way-delay observed during the experiment. So the absolute

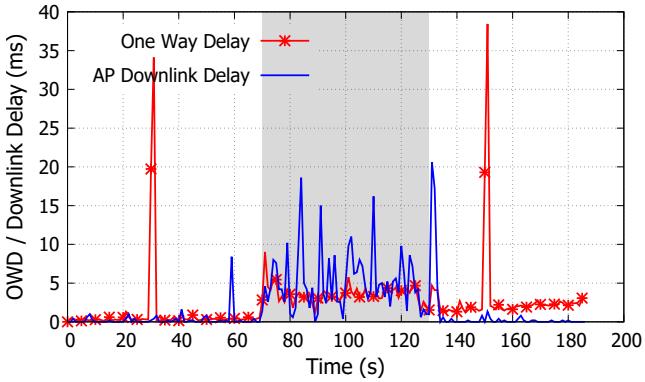


Figure 5: The One-Way Delay of a UDP flow and the AP downlink delay as observed by the `Ping-Pair` technique. The shaded area depicts the time period in which the second AP was congested causing interference in the first AP that the UDP flow was using.

values of one-way delay are not comparable to the downlink delay values estimated by `Ping-Pair`. Note also that interference from contenders outside our control results in spikes. However, the spikes are more prominent in the downlink delay time series because the sampling frequency (5 measurements per second) is much lower than for the one-way delay time series (50 samples per second, given the 20 ms inter-packet spacing); an averaging interval of 1 second is used for both time series.

Therefore, we conclude that the `Ping-Pair` technique is capable of accurately detecting Wi-Fi downlink delays caused by congestion and contention from traffic in the same AP as well as neighbouring APs via co-channel interference.

8.2 Channel Access Delay Estimation

Using our channel access delay estimator implementation in Linux, we conducted extensive experiments to evaluate its effectiveness. To create channel contention, we introduced additional channel contenders one by one to the network. These clients uploaded UDP packets to a server at the rate of one per millisecond. Another client, connected to the same AP, conducted channel access delay measurements. For each number of contenders, we gathered around 1500 channel access delay estimates. As expected, Figure 6 shows that the average channel access delay increases with the number of channel contenders in the network. The channel access delay estimated is high even with zero contenders because of contention from other nodes on the same channel which were not in our control. However, as shown in Figure 7, the channel access delay estimated for high-priority packets remains low (around 9 microseconds), whether there are any contenders or not. This points to the accuracy of our estimation technique.

8.3 Skype with Kwikr

We ran controlled experiments of `Kwikr`-enabled Skype to evaluate the effectiveness of the improved bandwidth adaptation mechanism. We also look at scenarios where `Kwikr`-enabled clients co-exist with

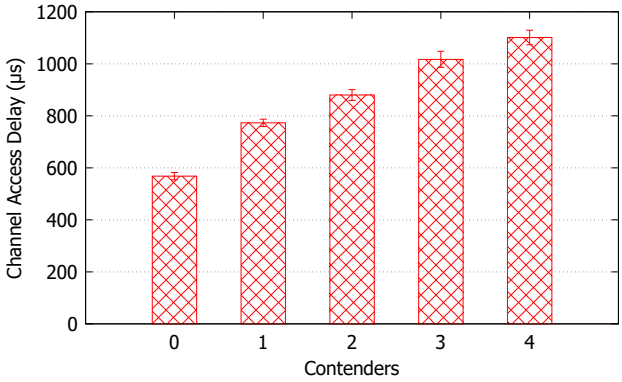


Figure 6: The estimated channel access delays for varying number of channel contenders. The error bars show the 95^{th}%ile confidence intervals.

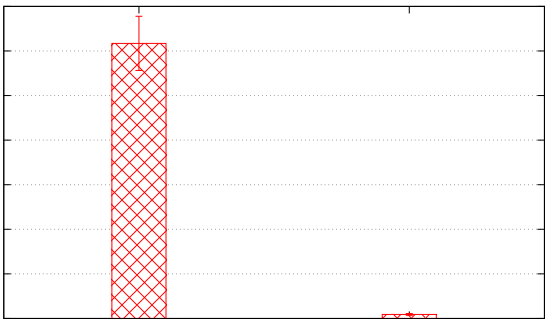


Figure 7: The estimated channel access delays for ping packets of normal priority and high priority. The error bars show the 95^{th}%ile confidence intervals.

non-Kwikr-enabled clients, showing that the improvement is not coming at the expense of other users.

Note that the experiments here are performed with Skype running on Android phones, whereas those in Section 3 were performed on Windows laptops. So the bandwidth numbers are not directly comparable because of differences in hardware and also in the specifics of the Android vs. Windows implementations of Skype.

We first show that Kwikr enables a better response to congestion. We ran 40 experiments, half with unmodified Skype clients and half that were Kwikr-enabled. We initiated three minute calls in each of these experiments. During the middle minute we introduced congestion in the form of heavy TCP flows by having two clients connected to the Wi-Fi link download data (using 20 parallel threads each) from a server connected to the wired link.

Figure 8(a) shows a representative execution; the shaded area represents congestion. Skype with Kwikr maintains a higher data rate during the congested period, as it has determined that Skype is not a significant cause of congestion. It recovers quickly when congestion ends.

Table 2: Kwikr flows co-existing with other flows. Table contains data rate for the measured flow when running simultaneously with the specified background flow.

	Background Flow	
Measured Flow	Skype	Skype with Kwikr
Skype	641 ± 7 kbps	616 ± 10 kbps
Skype with Kwikr	647 ± 9 kbps	652 ± 6 kbps

Figure 8(b) shows a CDF of the average data rate for each of the 40 calls, and we see that overall Kwikr enables a higher data rate. On average, the calls with Kwikr had a 20% higher data rate. We also see in Figures 8(c) and 8(d) that the round-trip time and loss are not hurt by Kwikr. (Instead of a CDF, for clarity here we simply note a few percentiles.)

Thus, we see that Kwikr-enabled Skype avoids backing off sharply in an overly conservative manner when congestion is due to cross-traffic, and also recovers more quickly. Note, however, that even with Kwikr enabled, Skype does back off. For example, this can be seen in the blue Skype with Kwikr curve in Figure 8a at around time T=65 seconds. It is just that Skype with Kwikr is not needlessly conservative.

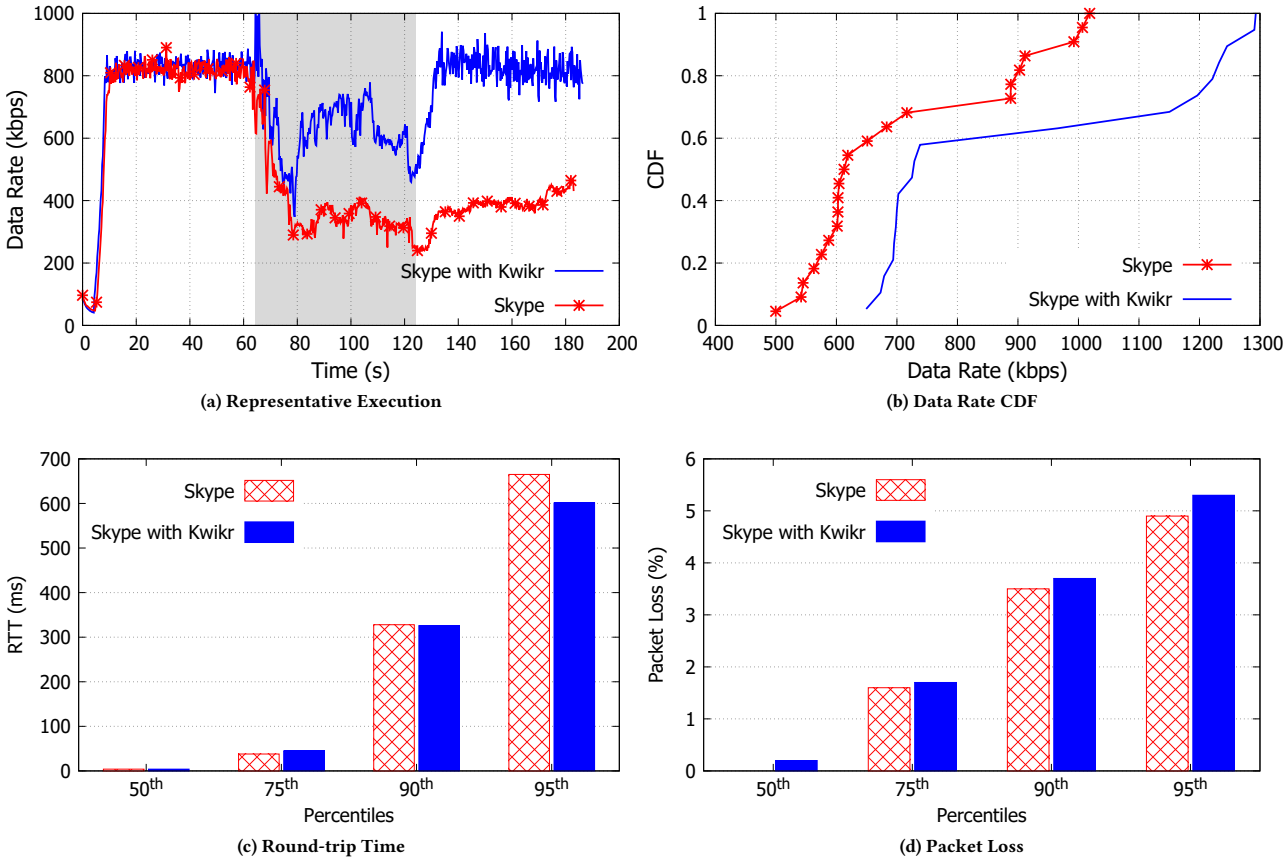
We also consider the scenario where the congestion is self-inflicted, i.e., where Skype really needs to back off sharply. We ran experiments where, on a regular AP, bandwidth was artificially throttled mid-stream using a token bucket filter during three minute calls with both regular and Kwikr-enabled Skype. Figure 9(a) shows that Kwikr does not affect bandwidth adaptation when congestion is self-inflicted.

One question is whether the apparent conservativeness of Skype with Kwikr during in this situation is due to the limited throttled capacity of the underlying link; in other words, it is that Skype with Kwikr remains aggressive but the link will not allow it to go any faster? If this were the case, the aggressive behaviour would be reflected in increased packet loss. However, as Figure 9(b) clarifies, the packet losses experienced with Kwikr is similar to that without Kwikr. This indicates that Skype with Kwikr is appropriately conservative since Ping-Pair allows it to correctly detect self-congestion.

Finally, we looked at the performance of Kwikr when co-existing with other clients. We ran 30 experiments where two clients made simultaneous two-minute calls. For ten calls, both clients were Kwikr-enabled; for another ten calls, one client was Kwikr-enabled and the other was not; for the final ten calls, both were unmodified Skype clients. Table 2 shows that co-existence does not have a significant impact on data rate. The non-Kwikr-enabled clients are essentially unaffected by co-existing with Kwikr clients. Similarly, Kwikr-enabled clients running together also appear to be unaffected.

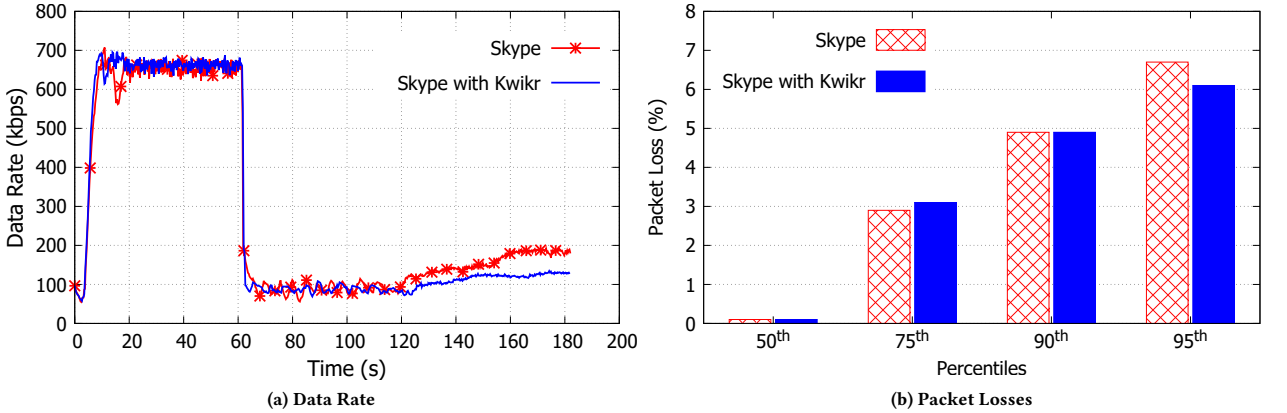
8.4 Results from Skype Production Release

Our implementation of congestion detection and bandwidth adaptation in Skype using Ping-Pair-based hints is part of the Android client in production and has been enabled for a fraction of the calls for limited periods to enable randomized A/B testing. (The goal is to eventually enable it for all calls once the testing has been



(a) Representative Execution (b) Data Rate CDF (c) Round-trip Time (d) Packet Loss

Figure 8: The `Ping-Pair` technique used to adapt to congestion. Figure (a) shows a representative execution showing how `Kwikr` responds more agressively to congestion (during the shaded interval in which the Wi-Fi network was congested). Figure (b) shows a CDF of data rate, showing the higher data rates achieved by `Kwikr`. Figures (c) and (d) shows that the round-trip time and packet loss are not hurt by the more aggressive adaptation strategy.



(a) Data Rate (b) Packet Losses

Figure 9: When congestion is self-inflicted, `Kwikr` reduces the data rate as in regular `Skype` (a). The losses experienced by the two flows are similar as well (b).

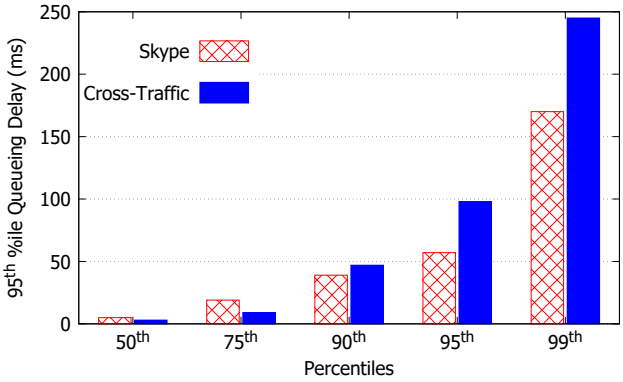


Figure 10: Wi-Fi downlink delay in the wild.

Table 3: Bandwidth gains due to `Kwikr` in deployment (with p-values showing statistical significance).

95%tile queuing delay due to cross-traffic (ms)	% calls covered	Avg. gain in bandwidth (p-value)	Median gain in bandwidth (p-value)
75	7.1%	3.3% (0.04)	3.1% (0.09)
100	4.6%	4.1% (0.05)	4.1% (0.08)
150	2.7%	7.4% (0.01)	8.6% (0.01)

concluded.) This has allowed us to take a peek into the prevalence of congestion in Wi-Fi networks in the wild (which, to our knowledge, has not been measured so far, in the absence of a technique such as `Ping-Pair`) and also report on the benefits to bandwidth adaptation from using `Ping-Pair`-based hints. We analyze a set of 119,789 video calls from the period Dec 9-16, 2016.

Figure 10 shows the distribution of Wi-Fi downlink delay, as measured by the Skype clients in production using `Ping-Pair`, due to Skype traffic and that due to cross-traffic. For each call, we extract the 95th-percentile queuing delay measurement (representing the worst 5% of the call) and plot the distribution of these 95th-percentile delays across the set of 119,789 calls. We observe that there is a significant amount of delay at the Wi-Fi downlink and moreover the delay due to cross-traffic dominates that due to Skype. For instance, for the worst 5% of calls, the queuing delay due to cross-traffic, at the 95th percentile, is at least 98 ms, whereas for the worst 1% of calls, it is at least 245 ms. To put these numbers in perspective, the end-to-end round-trip delay budget for VoIP is typically 300 ms, so for the worst calls, queuing at the Wi-Fi downlink alone takes a significant bite out of this budget.

The significant delay at the Wi-Fi downlink and the dominance of cross-traffic means that `Kwikr` can provide benefit by helping avoid unnecessarily conservative bandwidth adaptation. Table 3 presents the average and median bandwidth of calls, with and without `Kwikr`, for different degrees of cross-traffic-induced congestion at the Wi-Fi downlink. We see modest gains in average and median bandwidth of about 8% using `Kwikr`, with the gains being greater the larger the contribution of cross-traffic to queuing delay is.

Since we use the existing telemetry pipeline in Skype, we are unable to report bandwidth specifically from the congestion episodes. So we report average and median bandwidth over entire calls whose average duration is 967 seconds (just over 16 minutes), which results in the modest overall gains. However, to put these gains in perspective, we considered Skype traces gathered by us in controlled experiments and find that although the overall gains are modest here too, the gains during a short but still noticeable congestion episodes are high. For example, in one of our traces where we introduced congestion for about 30 seconds during a 3 minute call, the overall improvement in bandwidth was 9% whereas, during the congestion period the bandwidth gain was 21%. This shows that even though overall gains are modest there could be significant gains in shorter but noticeable periods.

Finally, note that the above bandwidth gains come with no statistically significant degradation in RTT or packet loss. This shows that `Kwikr` provides benefit while exhibiting safe behaviour towards the network.

9 CONCLUSION

We have presented `Kwikr`, a practical approach to improving bandwidth estimation for real-time streaming applications. Its centerpiece is a novel `Ping-Pair` technique to estimate delays at the Wi-Fi AP's downlink caused by congestion and contention, both in the same AP as well as neighbouring APs via co-channel interference. `Kwikr` yields bandwidth gains ("benefit") while remaining responsive to self-congestion ("safety"). Randomized A/B testing of `Kwikr` in the context of Skype, a popular AV conferencing application, shows that delays at the Wi-Fi downlink is significant and that bandwidth gains can be had without negatively impacting the network.

10 ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their valuable feedback on this paper. Baranasuriya and Gilbert were supported, in part, by the MOE AcRF MOE2014-T2-1-157 grant. This work would not have been possible without the help of Esbjorn Dominique, Sergey Galkin, Matej Kohut, Dariusz Marcinkiewicz, Georgios Panagiotou and Teemu Suutari from Microsoft Skype, who helped us to integrate our tool into the Skype consumer application on Android. The authors would also like to thank Vishnu Navda from Microsoft for discussions in the early stages of this work.

REFERENCES

[1] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. [n. d.]. Data Center TCP (DCTCP). In *SIGCOMM Computer Communication Review*.
[2] Anderson, Thomas E and Collins, Andy and Krishnamurthy, Arvind and Zahorjan, John. 2006. PCP: Efficient Endpoint Congestion Control.. In *NSDI*.
[3] Angelos Vlavianos and Lap Kong Law and Ioannis Broustis and Srikanth V. Krishnamurthy and Michalis Faloutsos. 2008. Assessing link quality in IEEE 802.11 Wireless Networks: Which is the right metric?. In *IEEE PIMRC*.
[4] A. Bakre and B. R. Badrinath. 1995. I-TCP: Indirect TCP for Mobile Hosts. In *ICDCS*.
[5] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz. 1996. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. In *SIGCOMM*.
[6] Hari Balakrishnan, Srinivasan Seshan, Elan Amir, and Randy H. Katz. 1995. Improving TCP/IP Performance over Wireless Networks. In *MobiCom*.
[7] Nimantha Baranasuriya, Vishnu Navda, Venkata N. Padmanabhan, and Seth Gilbert. 2015. QProbe: Locating the Bottleneck in Cellular Communication. In *CoNEXT*.

[8] Adam Bergkvist, D Burnett, and Cullen Jennings. 2012. WebRTC 1.0: Real-time Communication Between Browsers. *World Wide Web Consortium* (2012).

[9] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. 1994. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *SIGCOMM*.

[10] Ramón Cáceres and Venkata N. Padmanabhan. 1996. Fast and Scalable Handoffs for Wireless Internetworks. In *MobiCom*.

[11] Song Cen, Pamela C. Cosman, and Geoffrey M. Voelker. 2003. End-to-end Differentiation of Congestion and Wireless Losses. *IEEE/ACM Transactions on Networking* (2003).

[12] Dah-Ming Chiu and Raj Jain. 1989. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN systems* (1989).

[13] L. De Cicco, G. Carlucci, and S. Mascolo. 2013. Understanding the Dynamic Behaviour of the Google Congestion Control for RTCWeb. In *International Packet Video Workshop*.

[14] Mo Dong, Qingxi Li, Doron Zarchy, P. Brighten Godfrey, and Michael Schapira. 2015. PCC: Re-architecting Congestion Control for Consistent High Performance. In *NSDI*.

[15] Allen B. Downey. 1999. Using Pathchar to Estimate Internet Link Characteristics. In *SIGMETRICS*.

[16] Feng Lu and Hao Du and Ankur Jain and Geoffrey M. Voelker and Alex C. Snoeren and Andreas Terzis. 2015. CQIC: Revisiting Cross-Layer Congestion Control for Cellular Networks. In *ACM HotMobile*.

[17] Sally Floyd. 1994. TCP and Explicit Congestion Notification. *SIGCOMM Computer Communication Review* (1994).

[18] Sally Floyd and Van Jacobson. 1993. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking* (1993).

[19] Cheng Peng Fu and S. C. Liew. 2003. TCP Veno: TCP Enhancement for Transmission Over Wireless Access Networks. *IEEE Journal on Selected Areas in Communications* (2003).

[20] T. Goff, J. Moronski, D. S. Phatak, and V. Gupta. 2000. Freeze-TCP: A True End-to-End TCP Enhancement Mechanism for Mobile Environments. In *INFOCOM*.

[21] Mark Handley, Sally Floyd, Jitendra Padhye, and Jörg Widmer. 2002. TCP Friendly Rate Control (TFRC): Protocol Specification. (2002). RFC 5348.

[22] Toke Høiland-Jørgensen, Michał Kazior, Dave Täht, Per Hurtig, and Anna Brunstrom. 2017. Ending the Anomaly: Achieving Low Latency and Airtime Fairness in WiFi. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 139–151.

[23] S. Holmer, H. Lundin, G. Carlucci, L. De Cicco, and S. Mascolo. 2016. A Google Congestion Control Algorithm for Real-Time Communication. Internet draft (July 2016). URL https://tools.ietf.org/html/draft-ietf-rmcat-gcc-02.

[24] Ningning Hu, Li (Erran) Li, Zhuoqing Morley Mao, Peter Steenkiste, and Jia Wang. 2004. Locating Internet Bottlenecks: Algorithms, Measurements, and Implications. In *SIGCOMM*.

[25] V. Jacobson. 1988. Congestion Avoidance and Control. In *SIGCOMM*.

[26] Ingemar Johansson. 2014. Self-clocked Rate Adaptation for Conversational Video in LTE. In *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop*.

[27] S. J. Julier and J. K. Uhlmann. 1997. A New Extension of the Kalman Filter to Nonlinear Systems. In *AeroSense*.

[28] Partha Kanuparthy, Constantine Dovrolis, Konstantina Papagiannaki, Srinivasan Seshan, and Peter Steenkiste. 2012. Can User-Level probing Detect and Diagnose Common Home-WLAN Pathologies. *ACM SIGCOMM Computer Communication Review* (2012).

[29] Peggy Karp. 1971. DSCP - RFC 2474. (1971). Retrieved October 8, 2017 from https://tools.ietf.org/html/rfc2474

[30] Dina Katabi, Mark Handley, and Charlie Rohrs. 2002. Congestion Control for High Bandwidth-Delay Product Networks. *SIGCOMM Computer Communication Review* (2002).

[31] Rajat Kateja, Nimantha Baranasuriya, Vishnu Navda, and Venkata N. Padmanabhan. 2015. DiversiFi: Robust Multi-Link Interactive Streaming. In *CoNEXT*.

[32] Hyunwoo Nam Kyung-Hwa Kim and Henning Schulzrinne. 2014. WiSlow: A Wi-Fi Network Performance Troubleshooting Tool for End Users. In *INFOCOM*.

[33] Karthik Lakshminarayanan, Venkata N. Padmanabhan, and Jitendra Padhye. 2004. Bandwidth Estimation in Broadband Access Networks. In *IMC*.

[34] J. Postel. 1981. Internet Control Message Protocol. (1981). RFC 792.

[35] KK Ramakrishnan and Raj Jain. 1990. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks. *ACM Transactions on Computer Systems* (1990).

[36] Lenin Ravindranath, Calvin Newport, Hari Balakrishnan, and Samuel Madden. 2011. Improving Wireless Network Performance Using Sensor Hints. In *NSDI*.

[37] Christoffer Rodbro, Soren Andersen, and Koen Vos. 2009. Systems and Methods for Controlling Packet Transmission from a Transmitter to a Receiver via a Channel that Employs Packet Queuing when Overloaded. (2009). U.S. Patent Number 8259570.

[38] Sea Shalunov, Greg Hazel, Janardhan Iyengar, and Mirja Kuehlewind. 2012. Low Extra Delay Background Transport (LEDBAT). (2012). RFC 6817.

[39] Srikanth Sundaresan, Yan Grunenberger, Nick Feamster, Dina Papagiannaki, Dave Levin, and Renata Teixeira. 2013. WTF? Locating Performance Problems in Home Networks. (2013). Georgia Institute of Technology SCS Technical Report GT-CS-13-03.

[40] The Wireshark team. 2017. "Wireshark . Go Deep.". (2017). Retrieved October 8, 2017 from https://www.wireshark.org/

[41] E.A. Wan and R. Van Der Merwe. 2000. The Unscented Kalman Filter for Nonlinear Estimation. In *AS-SPCC*.

[42] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *NSDI*.

[43] Xinggong Zhang, Yang Xu, Hao Hu, Yong Liu, Zongming Guo, and Yao Wang. 2012. Profiling Skype Video calls: Rate Control and Video Quality. In *INFOCOM*.