Latency-Based WiFi Congestion Control in the Air for Dense WiFi Networks

Changhua Pei[†], Youjian Zhao[†], Yunxin Liu[§], Kun Tan[‡], Jiansong Zhang[§], Yuan Meng[†], Dan Pei[†*]
[†]Tsinghua University [‡]Huawei [§]Microsoft Research
[†]Tsinghua National Laboratory for Information Science and Technology (TNList)

***Abstract*— WiFi has become the primary method to access the Internet. However, the WiFi-hop latency, particularly in dense-WiFi environments, is far from satisfactory [1], to support delay-sensitive applications such as Web browsing and VoIP. The WiFi latency mainly comes from two kinds of queues: the host queue and the distributed queue, which is caused by CSMA/CA mechanism when multiple nodes contend for the channel. While the host queue can be easily bypassed using priority scheduling at end-host, the distributed queue is not. Previously, IEEE 802.11e tries to provide priorities in this distributed queue by adjusting the MAC layer parameters, but it does not scale when there are increasing number of delay-sensitive flows.**

In this paper, we propose and design QAir, a practical solution to reduce WiFi latency of delay-sensitive flows in dense WiFi networks. QAir takes a different approach to transfer this distributed queue to host queue. Consequently, the delay-sensitive flows can bypass the entire queue and their latency can be greatly reduced. QAir works in a distributed manner with no centralized scheduler. We have implemented QAir on commodity WiFi devices. Experimental results show that, compared to the 802.11 DCF baseline, QAir can reduce the average WiFi-hop latency of delay-sensitive flows by 50-75%.

I. Introduction

WiFi is one of the most popular methods to access the Internet. Although the data rate of WiFi keeps increasing quickly (*e.g.*, IEEE 802.11 ac provides a capacity of as high as 1300 Mbps), the MAC layer latency, however, remains high. This latency can be particularly long in *dense WiFi networks* where there are a large number of contending WiFi nodes. According to a recent measurement study [1] in a dense WiFi environment, the 90^{th} percentile of WiFi hop latency can exceed 25ms in the wild, making it hard to support delay-sensitive traffic such as online gaming and VoIP. Worse, application-level metrics, *e.g.*, page load time in Web browsing, may be $100\times$ magnified by the last-hop latency [2], due to multi-round data transmissions and complex application-level logics. Thus, a 25ms WiFi latency may lead to a page load time of 2500ms, resulting in unacceptable quality of user experience (QoE).

The fundamental reason behind this large latency can be explained as follows. Each WiFi frame will experience two queues. The first queue is the *local host queue*, which includes every queue from application layer, IP, 802.11 driver (*e.g.*, *ath9k*), and physical layer (PHY) queue. Once a frame reaches the HOL of PHY queue, the DCF mechanism kicks in, and the frame is now in a second queue, the distributed queue in the air (*i.e.*, wireless channel) where all HOL frames at all nodes wait in a line for accessing the channel. Delay-sensitive flows can potentially bypass the local host queue with priority scheduling [3], but they cannot bypass the distributed queue in the air (shared channel) which is imposed by DCF (Distributed Coordination Function) [4] in the 802.11 CSMA/CA contention protocols.

The distributed queue in DCF roughly works as follows. Suppose a WiFi frame is at the head of line of PHY queue of a host, the wireless node (node A) will sense whether it is at the head of line of the distributed queue (*i.e.*, channel is idle for a DIFS interval). If so, the frame is sent out to the air; otherwise, the frame is placed in the distributed queue at a location determined by a random value in [0,CW], where CW is the Contention Window. Every time slot, node A senses the channel again, and, if the channel is idle, moves the frame one step forward towards the head of distributed queue. If a frame is at the HOL of the distributed queue (*i.e.*, when timer expires), node A sends it out to the air. If collision happens, node A will place the retry frame in a location further back in the distributed queue (*i.e.*, doubling the contention window CW). Note that in a dense network, the distributed queue length is heavily influenced by the number of contending nodes. In general, the more contenders, the longer the distributed queue and the higher the collision rate (when multiple nodes think they are at HOL of the distributed queue), thus the longer the latency.

Therefore, the fundamental problem is how to reduce the latency caused by the distributed queue, where host-level priority does not help. EDCA (Enhanced Distributed Channel Access) in IEEE 802.11e allows frames with higher priority tags, which have smaller contention parameters (CW and IFS), to wait a shorter time in the distributed queue. However, as the density of WiFi networks and the number of delay-sensitive flows keep increasing, more and more flows will be assigned into the highest priority category. This might cause serious collision problems. To keep up with the increasing number of delay-sensitive flows, IEEE 802.11e has to increase the CW of each priority class, but a large CW will cause high protocol cost on the channel which will further decrease the overall throughput. In one word, the way by assigning different flows different CW is not practical in today's dense WiFi environment.

* Dan Pei is the corresponding author.

978-1-5386-2704-4/17/$31.00

Our key idea to address the long latency problem caused by the distributed queue is the following. Ideally, suppose there is a centralized scheduler that globally schedules a frame's delivery to the PHY queue on each host. Such a scheduler can control the length of the distributed queue (thus the latency of the a HOL frame in PHY queue) by regulating the frame delivery rate from the driver to the PHY queue. The scheduler can also pick the frame with the highest priority (which might actually arrive the latest) in the driver queue to deliver to PHY queue. Thus a delay-sensitive frame can get the highest priority locally (up to driver queue) and globally benefit from a much smaller distributed queue. Unfortunately there is no centralized scheduler, and the key challenge now is to develop an algorithm that achieves similar effects but runs on nodes in distributed manner.

We tackle the above key challenge using the following mechanisms: 1) each node measures the per-frame latency to estimate the number of contenders on the channel; 2) a frame to be sent into the physical layer waits in the driver for a time interval that is a function of the estimated number of contenders. In other words, the length of distributed queue can now be roughly controlled (*e.g.*,1) instead of being as large as the number of contenders. The waiting time happens mainly at the local queue, and late-arriving but high priority frames can now be put to the front of the local queue in the driver (*i.e.*, bypassing majority of the host queue, excluding the PHY queue) and quickly be sent to the distributed queue. The latency measurement and the control logic are analogous to the RTT measurement and congestion control in TCP (see §III for more details). We thus name our proposed approach as **QAir** (congestion control for the **Q**ueue in the **Air**).

QAir is a very practical solution and easy to be deployed in existing WiFi networks. The control algorithm works in a distributed way on each device, without requiring a centralized scheduler. The changes to existing devices are minor and all in software. Each device only needs to measure a latency and implement the control algorithm, which can be easily done by a software patch. QAir does not require but can work with explicit priority tagging, and also provides a practical implicit priority for delay sensitive applications.

We have implemented QAir on commodity WiFi devices and built a dense WiFi testbed to evaluate its performance. Experimental results show that QAir is able to reduce $ping$ RTT by 50% to 75% and without sacrificing throughput performance, compared to the baseline IEEE 802.11 DCF mechanism. In another small scale personalized live video streaming experiment, QAir reduces the number of buffering events from 7 to 0 and the e2e delay from 9.78s to 2.25s.

We make the following contributions in this paper:

- We propose and design a novel approach to reduce the latency of WiFi without sacrificing throughput. Our approach is practical and easy to deploy (§III and §IV).
- We implement our approach on commodity devices and demonstrate how to accurately measure the delay of frame in physical layer (§V).

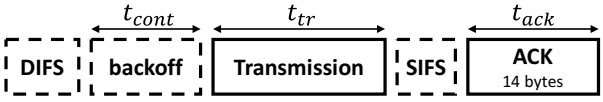


Fig. 1: Breakdown of per-frame latency in IEEE 802.11.

TABLE I: IEEE 802.11 parameters. GI means guard interval.

Variant	Bit rate(Mbps)	DIFS,SIFS,SLOT(us)
802.11n	400ns GI: 15-150	2.4GHz: 28/50, 10, 9/20
	800ns GI: 13.5-135	5.0GHz: 34, 16, 9
802.11ac	400ns GI: 65-866.7	34, 16, 9
	800ns GI: 58.5-780	

- We conduct experiments to evaluate our implementation and verify the effectiveness of our approach (§VI).

II. WiFi Latency and Related Work

In this section, we define the problem of per-frame latency in WiFi and describe the related work.

A. Per-Frame Latency in WiFi

In this paper, we aim to reduce the latency of a network packet (*i.e.*, a frame in WiFi MAC) imposed by WiFi MAC layer. Such a latency is the time period between the time when a frame is selected to be sent out by the WiFi network interface card (NIC) and the time when the frame is successfully sent out. Note that it does not include the queuing time of the frame in the WiFi NIC [1].

Figure 1 illustrates the breakdown of a frame transmission. The per-frame latency consists of two parts: a fixed part including the DIFS, SIFS, the transmission of the frame (t_{tr}), and the transmission of the ACK (t_{ack}); and a dynamic part of the backoff time (t_{cont}). t_{cont} is the latency caused by the CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) mechanism of WiFi. In CSMA/CA, to avoid transmission collision, a WiFi device waits for a time period randomly selected from a contention window (CW) of $[0, CW]$ before transmitting a frame. The initial value of CW is $CWmin$.

Even with the CSMA/CA mechanism, collisions may still occur, when there are multiple devices in the same WiFi network. If a collision happens, the frame transmission is failed. The contention window will be doubled (capped by a value of $CWmax$) and the frame must be re-transmitted. In the case of multiple devices competing with each other to access the channel, per-frame latency (labeled as t) can be modeled by the following equation:

$$t = t_{const} + t_{cont} + t_{tr} \quad (1)$$

where t_{const} is a constant time one frame must spent according to IEEE 802.11 protocol. $t_{const} = DIFS + SIFS + t_{ack}$, which can be calculated using the values of Table I. t_{tr} can

[1]However, for evaluations in §VI, we use the overall latency including all the queuing time which is perceivable to users.

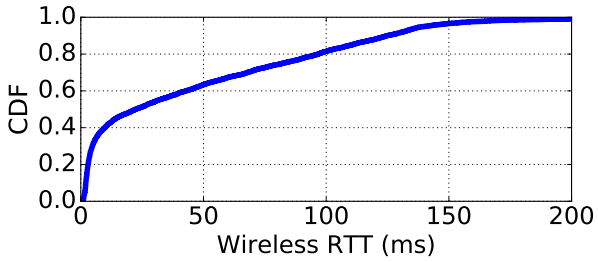


Fig. 2: The WiFi-hop latency in the wild.

be calculated using the frame size and physical transmission rate of this frame: $t_{tr} = framesize/phyrate$.

The rest part is the contention time t_{cont}. It is a random value from 0 to t_{cont}^{max} which can be calculated using the following equation:

$$t_{cont}^{max} = SLOT * CW_{max} + \sum_{1}^{\beta-1} \frac{framesize}{phyrate}$$

where β is the number of contenders. $\sum_{1}^{\beta-1} \frac{framesize}{phyrate}$ is the maximum waiting time for other $\beta - 1$ devices finish their transmissions.

When we put all the above components together, we can get Equation 2.

$$t^{max} \approx t_{const} + SLOT * CW_{max} + \beta * \frac{avgframesize}{avgphyrate} \quad (2)$$

The above equation shows that the per-frame latency ($t \in [t_{const}, t^{max}]$) increases linearly with the number of contenders (β). This is because the more contenders, the longer the distributed queue and thus the larger the latency.

To study the WiFi latency in the real world, we conducted an experiment to measure the WiFi-hop latency in a classroom. We used a WiFi client to *ping* the associated AP (Access Point) to exclude the latency of the wired part. We put the client near the AP to avoid interference (*e.g.*, hidden terminal problem). We use our rouge AP with only one client associated on it to exclude the effects of local contention at the AP side. As shown in Figure 2, 40% of packets had a WiFi hop latency (one-way) larger than 25ms, which is even worse than a recent measurement study [1].

Motivated by the large WiFi latency in the real world, we seek for a practical solution that is easy to deploy in existing WiFi networks to solve the high-WiFi-latency problem. The targeted scenario in this paper is the "high density" scenario where multiple devices compete with each other with mixed types of traffic. Some devices have large volume traffic that may not be delay sensitive but may eat all the available network capacity, while other devices have low-volume but delay sensitive traffic. Our goal is to reduce the WiFi latency, particularly for delay sensitive traffic, but without sacrificing the overall network throughput.

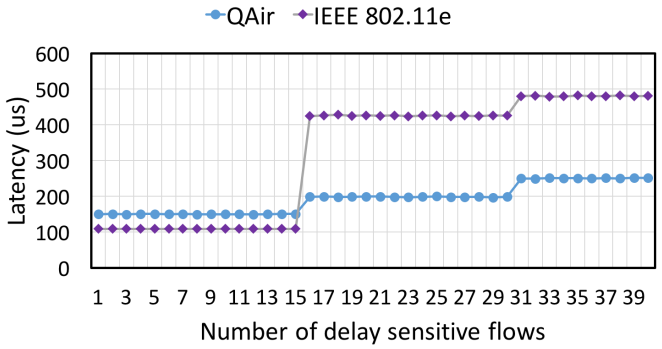


Fig. 3: When there are multiple voice flows, collision probability will be very high in IEEE 802.11e that significantly hurts overall latency.

It is important to note that "easy-to-deploy" is the primary goal of our work and it differentiates our approach from the existing ones. As we will describe below, existing solutions cannot or are hard to be used in the real world.

B. Related Work

We classify related work into three categories according to their optimization goal: IEEE 802.11e and central scheduling aiming to reduce the length of distributed queue, end-host based QoS (Quality of Service) solutions aiming to do priority scheduling of host queue, and other work aiming to improve the throughput by contention control.

IEEE 802.11e: From 802.11e-2005 amendment, the standards have incorporated four priorities for four traffic categories: voice, video, best effort and background. By assigning different contention parameters,*i.e.*, inter-frame spacing (IFS) and contention window (CW), higher priority traffic can gain earlier access during contention.This approach, called enhanced distributed channel access (EDCA) in the standards or wireless multimedia (WMM) as advertised, has been implemented in many WiFi devices. In literatures, there are a lot of work on performance evaluation [5], [6], [7], [8], [9] and improvement[10], [11] for EDCA. Moreover, [12], [13], [14] propose improvements for differentiated services using EDCA.

However, as has been pointed out before [15], in today's dense WiFi network, performance issue will out-weight the benefit of 11e prioritization. Specifically, 11e assigns very small contention window to voice traffic which is $1/4$ of the default. When there are multiple voice flows, collision probability will be very high that significantly hurts overall latency and throughput [15], which is shown in Figure 3. Compared to 11e, QAir does not need small contention window therefore will not suffer from high collision. As more and more traffic becomes sensitive to latency and desires high priority, *e.g.*, instant messaging, live streaming, web browsing, *etc*, things will get worse.

Centralized Scheduling: The representative approach is the Point Coordination Function (PCF) or Soft-TDMAC[4] in the

TABLE II: The comparison of the related works and QAir .

	Addressing distributed queue	Adapt to high density network	Central controller	Reduce latency
PCF,[4]	Yes	No	Yes	Yes
IEEE 802.11e	Yes	No	No	Yes
End-host QoS[3]	No	Yes	No	Yes
Idle Sense[25]	Yes	No	No	No
QAir	Yes	Yes	No	Yes

IEEE 802.11 standard. In PCF, the Access Point (AP) in a WiFi network coordinates all the communications within the network and thus there is no collision. Tan *et al.* proposed a scheme similar to PCF by placing a regulator above the MAC layer in an AP to provide equal time shares to client devices [16]. CENTAUR[17] and DOMINO[18] focus on building hybrid MAC protocols for central scheduling. All this kind of central scheduling solutions need a central controller and stringent time synchronization, making them not practical in practice. In fact, PCF is rarely implemented in consumer WiFi products. [19] aims to achieve the performance of centralized scheduling using a distributed manner. However, it still needs modifications in IEEE 802.11 PHY layer which makes it unpractical to be widely deployed in commodity routes. Today's commercial APs integrate PHY layer into wireless chip and normal users have no access to the PHY layer.

End-host Qos. Work like [3] aims to implement QoS solution on single end-host and is complementary to QAir because QAir mainly focuses on queuing in the air, *i.e.*, QoS across different nodes.

Contention Control. Many work [20], [21], [22], [23], [24] aims to improve aggregated throughput. These works help little on per-frame latency which is crucial to delay-sensitive flows. For example, Idle Sense [25] proposes a scheme to control contention level of a WiFi network without requiring a centralized controller. It can reduce collision rate and improve faireness and overall goodput, but it does not directly apply to latency reduction for delay sensitive flows.

Table II shows the comparison between existing solutions and our approach from multiple aspects.

III. QAir Overview

Figure 4 shows the architecture of QAir. QAir has three key components: Delay Monitor, Control Algorithm, and Rate Regulator. The Delay Monitor keeps measuring the delay of each frame and sends the measured delay to the Control Algorithm. Based on the received per-frame latency information, the Control Algorithm computes a maximum delivery rate and passes the rate to the Rate Regulator. The Rate Regulator

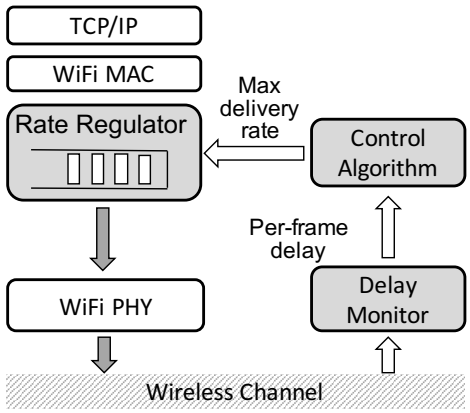


Fig. 4: The architecture of QAir.

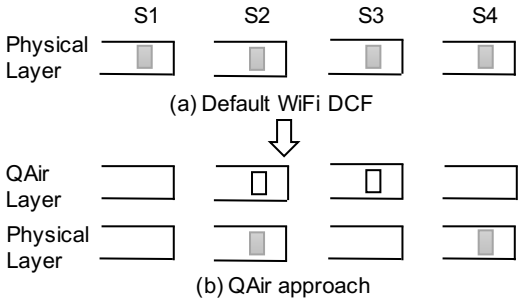


Fig. 5: QAir reduces per-frame latency by reducing the number of contention devices.

works between the 802.11 driver layer (*ath9k*) and the WiFi physical layer. Using the maximum delivery rate, it controls how many packets should be sent to the WiFi physical layer. If the traffic volume received from the *ath9k* layer is larger than the maximum delivery rate, the Rate Regulator buffers the extra packets instead of sending all the packets to the WiFi physical layer.

Figure 5 uses an example to illustrate how QAir is able to reduce WiFi latency. Assume there are four client stations (S1, S2, S3, and S4) in a WiFi network, each with one packet to send. In the WiFi DCF, all application packets are directly sent to the WiFi NIC [2] (*i.e.*, Physical Layer) which has a buffer to store the packets before they are successfully sent out. Thus, all the four stations have one frame in their NIC, competing with each other to access the wireless channel. Equivalently, they form a distributed queue of four frames. The per-frame latency is determined by four contention stations according to Equation 2. In QAir, with the rate regulation, two stations (S1 and S3) buffer their packet in the QAir layer and only the other two stations (S2 and S4) have a frame to send in their NIC. Consequently, the length of the distributed queue is halved, the number of contention stations is reduced from 4 to 2, and thus the per-frame latency is reduced according to Equation 2. Note that the packets buffered by QAir will have an extra queuing delay in the QAir layer. However, with less contention, the overhead of the backoff mechanism in WiFi is

[2]Unless the buffer on the NIC is full.

reduced and thus the overall delay of the packets still becomes smaller.

The example in Figure 5 is extremely simplified for illustration. In normal cases, QAir may buffer multiple packets and the WiFI NIC of a station may not always have only one frame. However, through rate regulation, QAir reduces the equivalent number of contenders and thus the overall contention level of the WiFi network is reduced, which not only reduces per-frame latency but also may improve the aggregated throughput of the WiFi network, as we will show in §VI.

In QAir, each station runs the control algorithm in a distributed way but all stations will converge to the same maximum delivery rate for a fair share of the wireless channel. If a station has more traffic than its fair share, its packets will be buffered by QAir and have a longer delay, while the packets of a station with less traffic than its fair share will be directly sent to WiFi Physical Layer without such an extra buffering delay. As a result, a station with a small flow will have a smaller delay than a station with a large flow [3]. In practice, delay-sensitive applications such as Web browsing and VoIP usually have small flows than delay-insensitive applications such as file downloading and video streaming. Therefore, QAir provides an *implicit* high priority for those small and delay-sensitive flows in practice.

By changing the parameters in the control algorithm, QAir is able to control the number of concurrent contenders for a flexible trade-off between latency and throughput. In the next Section, we describe how each of the key components of QAir works in details.

IV. Key Components of QAir

As shown in Figure 4, QAir uses a control loop of three steps to reduce WiFi latency: 1) measure the contention level of the channel, 2) decide the maximum delivery rate, and 3) regulate traffic to control the contention level of the channel. Next we describe how each step works.

A. Measuring Channel Contention

QAir measures the contention level of a channel by measuring per-frame latency. However, due to the random backoff mechanism in WiFi MAC and the channel dynamics, latency of frames may vary in a large range, making it a challenging task to get a stable measurement on the channel contention. Moving average is a widely-used technique to smooth out short-term fluctuations in time-series data [26]. In our case, the moving average can be expressed using Equation 3. Every time there is a new frame k sent out, its delay t_k will be used to calculate the new value of the measurement T. The weight parameter α is used to decide the weight of the old value of T in calculating the new value of T.

$$T(new) = \alpha * T(old) + (1 - \alpha) * t_k \tag{3}$$

[3]Here we assume that each station has only one flow but this assumption can be easily removed by maintaining multiple queues, each for a single flow.

However, the maximum frame delay $t_k(max)$ and the minimum frame delay $t_l(min)$ can have a difference of hundreds of times. If we directly use the moving average on each frame, the measured T still has large fluctuations. For example, under a same extent of contention, t_k can vary from 1ms to 100ms. Even we only give 10% weight (*i.e.*, $\alpha = 0.9$) to the latest frame, the final value of T will still have a variance as large as $10ms$. To address this issue, we take a two-step approach in calculating the value of T. First, we group frames into different time slots. In each time slot i, we calculate T_i as the arithmetic average of the latency of all the frames $(1, 2, ..., N)$ in slot i. Then, we use the moving average of T_i to decide the final measurement $\hat{T}$ across different time slots.

$$T_i = \frac{1}{N} \sum_{k=1}^{N} t_k \tag{4}$$

$$\hat{T}(new) = \alpha * \hat{T}(old) + (1 - \alpha) * T_i \tag{5}$$

The time slot size $checkInterval$ is a critical parameter. A large time slot cannot reflect the dynamics of the channel contention while a small time slot cannot eliminate the variance of T_i effectively. We recommend that $checkInterval$ should be set with an appropriate value so that a station can send at least X packets out in a time slot. Here X should be a comparable value of the minimum value of contention window, *i.e.*, $CWmin$. For instance, under the IEEE 802.11n standard, the maximum throughput is 600Mbps, $CWmin = 15$, maximum packet size is 1500 bytes, then $checkInterval = 15 * 1500 * 8/600 = 0.3ms$. However, in the real world measurement, the achievable throughput is less than 100Mbps, then $checkInterval$ should be larger than $1.8ms$.

B. Control Algorithm and Control Target

The pseudo code of our control algorithm is shown in Algorithm 1. The main functionality of the algorithm is to convert the input t_k into the control metric R_{tc}. t_k is the measured latency of one frame spent in the WiFi NIC as shown in Figure 1. t_k includes the transmission time t_{tr} and the backoff time t_{cont}. R_{tc} is used to as the *maximum delivery rate* to control how fast QAir should send packets to WiFi physical layer.

From line 5 in Algorithm 1 we can see that R_{tc} is adjusted every $checkInterval$. However, the procedure $F(t_k)$ is called every time a wireless frame is successfully transmitted. Lines 2-6 aim to average the per-frame latency t_k into the average metric $\hat{T}$ to eliminate the delay variance of the frames and reflect the extent of channel contention. Lines 11-14 are the control routine which is inspired by existing TCP-based congestion control algorithms like [27]. T_{target} is the control target we want the algorithm to converge to.

The control routine is simple but carefully designed. Once the averaged frame delay $\hat{T}$ is larger than the control target T_{target}, it means that the channel is too congested and each node reduces the R_{tc} with a factor of $\frac{T_{target}}{\hat{T}}$. However, R_{tc} cannot be smaller than the lower bound R_{min}. If $\hat{T}$ is smaller

than T_{target}, it means that the channel is not fully utilized and each node increases the R_{tc} by a step of δ. Similarly, R_{tc} cannot be larger than the upper bound R_{max}.

The control algorithm borrows the "Additive Increase and Multiplicative Decrease" principle from the TCP congestion control mechanism[28] and makes some modifications according to the characteristics of the wireless network. There is a fundamental difference of converge path between wireless MAC and wired TCP. In the wired scenario, once congestion happens, all the wired senders will know the congestion immediately and thus limit the rate accordingly. Then, the wired path is free of congestion and each wired sender can increase their sending rate. This procedure will continue many times until converging. All the senders keep one same pace in the wired scenario. In wireless MAC scenario, we use whether $\hat{T} > T_{target}$ to judge whether there are higher contention on the channel than we want. The difference is that not all stations sense the contention at the same time to reduce their rate (R_{tc} in our algorithm) in the wireless scenario. Sometimes one station A senses $\hat{T} > T_{target}$ and reduces the R_{tc}, it may still get $\hat{T} > T_{target}$ at the next time window $checkInterval$ and continues reducing the R_{tc}. At the same time, another station B may get $\hat{T} < T_{target}$ in both these two time windows because it beats station A.

Algorithm 1 Decide maximum delivery rate.

t_k : measured delay of frame k.
T_{target} : the delay constraint.
R_{tc} : *maximum delivery rate*.

procedure F(t_k)
  $sum \leftarrow sum + t_k$
  $ntrans \leftarrow ntrans + 1$

  if $(now\_time - last\_time) > checkInterval$ **then**
    $\hat{T} \leftarrow \alpha * \hat{T} + (1 - \alpha) * \frac{sum}{ntrans}$
    $last\_time \leftarrow now\_time$
    $sum \leftarrow 0$
    $ntrans \leftarrow 0$

    if $\hat{T} < T_{target}$ **then**
      $R_{tc} \leftarrow minimum(R_{tc} + \delta, R_{max})$
    else
      $R_{tc} \leftarrow maximum(\frac{T_{target}}{\hat{T}_i} * R_{tc}, R_{min})$

  return R_{tc}

To adapt to these wireless characteristics, we use R_{min} and R_{max} to bound the control metric R_{tc}. The lower bound R_{min} is mainly used to prevent the stations R_{tc} from decreasing to 0. Otherwise no frames will be sent and the procedure F(t_k) will never be called. The R_{tc} will remain 0 all the time. We call this the "dead corner" of the algorithm. The upper bound R_{max} is the maximum theoretical throughput one can achieve under certain IEEE 802.11 protocol. It mainly used to prevent the control metric from becoming too large, in this case, it will take more time to converge. For example, assume there are 2 stations (A and B) contending the channel at the stable state. Once station B finishes the transmission and leaves the contention, station A will find $\hat{T} < T_{target}$ all the time. Then, station A will keep increasing R_{tc} towards infinity and it will take too much time to converge when another station C joins the contention.

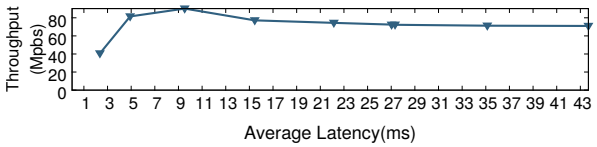


Fig. 6: The measured delay-throughput curves in different numbers of contenders (N) in the wild. The data points on the curve are N=1 to N=9 from left to right.

Control target. The control target T_{target} is the targeted per-frame latency that we want to achieve in a WiFi network. However, there is a fundamental trade-off between the per-frame delay and the aggregated throughput in WiFi protocols. Comprehensive theoretical analysis has been conducted on the trade-off in prior research and we also conduct measurement in the wild to decide a proper T_{target}. For example, Figure 6 plots the delay-throughput relation in different numbers of contenders under the IEEE 802.11n protocol from the measurement in the wild. We can see that if there are too few stations, *e.g.*, 1 or 2, the delay is very small but the throughput is also low. Therefore, to avoid sacrificing the throughput, we cannot set T_{target} to a too small value. Fortunately, there is a sweat point to balance the delay and throughput in the curve. For example, in the curve of $N = 3$, the point of the highest throughput still provides a small delay. According to Equation 2, by changing the value of T_{target}, we can change the number of concurrent contenders and thus provide a flexible trade-off between the latency and aggregated throughput.

C. Traffic Regulation

The work of the Traffic Regulator is simple: based on the given maximum delivery rate R_{tc}, it controls the speed of sending packets to the WiFi physical layer. We choose to use packets/second rather than bytes/second as the unit of R_{tc}. That is, we control how many packets rather than how much traffic volume to send to the physical layer. This is consistent with the DCF mechanism where each station has the same opportunity to send one frame, no matter what the frame size is. Whether a frame can get the chance to be sent out depends on the value of its random counter rather than the size of the frame.

Convergence. Our control algorithm is convergent. For each adjusting time window, we can sort the stations by their average frame delay $\hat{T}$. If all the stations' $\hat{T}$ is smaller than

T_{target}, they will all increase their control metric R_{tc} to send more frames. In this way, the average frame delay will certainly increase. If all the stations' $\hat{T}$ is larger than T_{target}, it indicates heavy congestion and each station will reduce its R_{tc} by a factor of $\frac{T_{target}}{\hat{T}}$. If some stations' $\hat{T}$ are larger than T_{target} while the others' $\hat{T}$ are smaller than T_{target}, the stations whose $\hat{T}$ is larger than T_{target} will decrease their R_{tc} by a factor of $\frac{T_{target}}{\hat{T}}$. The others will increase their R_{tc} by a fixed step δ. If the channel congestion does not reduce, it means there are more stations whose $\hat{T}$ is larger than T_{target}, *i.e.*, more stations begin decreasing their R_{tc} with a larger factor of $\frac{T_{target}}{\hat{T}}$. As the overall increasing-speed decreases and the overall decreasing-speed increases, the decreasing speed of R_{tc} will be larger than the increasing speed and the extent of contention will be mitigated. The closer the $\hat{T}$ to T_{target}, the smaller the adjustment. From the above analysis, our control rule is a closed loop and will eventually converge to a stable state.

Fairness. The fairness of QAir is based on the fairness of the DCF where each contender has the same probabilistic opportunity to send frames.

From Figure 1, we can conclude that all the stations's t^{max} (Equation 2) are the sum of own packets' transmission time and all the neighbor packets' transmission time. As the IEEE 802.11 DCF protocol ensures the packet level fairness for different flows, they get the same $\hat{T}$ in the long run. Furthermore, they have the same possibility to increase or decrease the control metric R_{tc} and will have the same R_{tc}.

Aggregation/MSDU: If the aggregation is enabled, multiple packets may be aggregated into one big packet (MSDU). In our measurement, we can distinguish these packets and regard all the small packets (MPDU) in the same MSDU as one, thus the aggregation is transparent to our QAir algorithm. Unless otherwise mentioned, we enable the aggregation mechanism in our following experiments.

V. Implementation

We have implemented QAir on OpenWrt-based routers by modifying an open source wireless driver ath9k that is widely used on many commodity routers. The modifications rely on only the basic callback functions of the driver and thus may be easily applied to other WiFi devices (*e.g.*, Android devices) if the wireless driver source is available.

The commodity routers we used for both implementation and evaluation are the NETGEAR WNDR 4300 dual band routers. They run OpenWrt version 15.05 and are equipped with 560MHz CPU, 128MB RAM and 128MB Flash. They may work as an AP or a client and thus we are able to setup a flexible testbed using only the routers.

Measuring per-frame latency. One challenge in our implementation is how to accurately measure per-frame latency on the routers. Similar to many other WiFi devices, the routers have a queue to buffer multiple frames in their WiFi NIC. Thus, we cannot directly decide when a frame is moved to the head of the queue to start trying to access the channel. To address the challenge, we take the following approach with three important timestamps for each frame: t_w, t_h and t_e. t_w is the time when the frame enters the hardware queue and waits in the queue. t_h is the time when the frame reaches the head of the queue and begins the DCF procedure. t_e is the time when the packet is successfully acknowledged by the MAC layer ACK and is removed from the queue. t_w can be directly recorded in the driver. It is also easy to measure t_e because the hardware will generate an interrupt to the driver once a frame is successfully acknowledged and removed from the hardware. t_h is hard to measure because due to the unknown waiting time of the frame in the queue.

Our approach is to infer the t_h of a frame from the t_e of the previous frame [29]. Assume we have two frames p and $p+1$ which are continuously added to the hardware queue in times of $t_w(p)$ and $t_w(p+1)$, the completion time of DCF for these two frames are $t_e(p)$ and $t_e(p+1)$. Then, we can get the accurate $t_h(p+1)$ using Equation 6. If $t_e(p) < t_w(p+1)$, it means that when frame $p+1$ is added into the hardware queue, frame p is already removed from the hardware queue, then frame $p+1$ will immediately reach the head of the queue and begin DCF procedure. In this case, $t_h(p+1) = t_w(p+1)$. Otherwise, frame $p+1$ has to wait for frame p being sent out before starting DCF, *i.e.*, $t_h(p+1) = t_e(p)$.

$$t_h(p+1) = MAX(t_e(p), t_w(p+1)) \tag{6}$$

Traffic Shaping. We use the token bucket algorithm to shape the traffic between the QAir layer and the physical layer to a desirable rate. Depending on whether there are enough tokens in the bucket or not, we decide whether to buffer a packet in the QAir layer or send it to the physical layer. The number of the tokens in the bucket normally represents the number of frames can be sent now. If one frame is transmitted, one token is removed. So the traffic will be shaped as the rate how fast the tokens are added into the bucket. We record the last time when the token is added. Every time when the R_{tc} is updated or when a new packet arrives, we will use the time gap and R_{tc} to calculate the tokens to be added. Consequently, the traffic will be shaped just as the R_{tc}. In our implementation, we add our entry function to where the ath_tx_txqaddbuf() is called and redirect packets to our token bucket. We use the *hrtimer* in the Linux kernel API, which has a precision of *micro second* level, to resume sending the packet after a appropriate time. All the packets will be delivered by ath_tx_txqaddbuf() for the next-step processing in the hardware.

VI. Evaluation

We implemented QAir on commodity routers, and use a 20-node testbed to evaluate QAir's performance in the wild.

A. Testbed Setup

Our testbed consists of 20 commodity routers (NETGEAR 4300 dual band) with QAir implemented, with a topology shown in Figure 7. All the nodes are connected via Gigabit Ethernet switch to one of the two commodity Dell R410

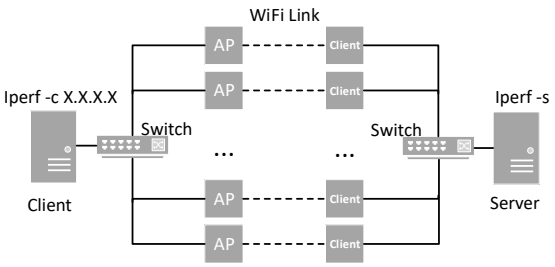


Fig. 7: Topology of the testbed.

servers equipped with 24 2.4GHz cores, 30 Gigabit memory, and Linux OS. Half of the routers are configured as Access points (APs), each of which as a client connected to. We configure the routing table on the Linux servers to send packets into the certain wireless links. Therefore, there are in total 10 AP-client links. This topology was intentionally setup as a dense one: all nodes are running on the same channel (802.11n, 5GHz band, 20MHz width), where there are no other nodes and contention/interference from the surrounding environment, and they are physically located close to each other so that all nodes are within contention range of any other node.

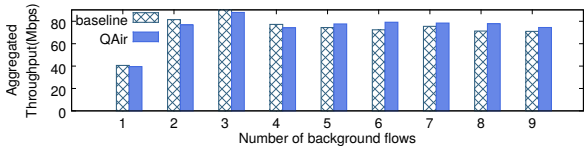
QAir can be turned on or off in user-space via configuration, and we compare the baseline case ("off") and QAir case ("on") in the following experiments.

It is noteworthy that in the real world, there may be multiple clients associated with the same AP. However, as QAir is an algorithm to improve the performance of different senders, it only care about the actual number of sender (or contenders) on the channel. It is transparent for QAir whether one sender's packets are heading for different nodes or multiple senders (clients) sending to the same node (AP). The only thing we need to do is to experiment different numbers of contenders to get comprehensive measurement results. Without loss of generality, we use one-to-one WiFi pair to evaluate the performance of QAir in a more intuitive way.

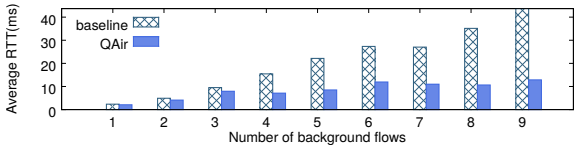
B. Performance Under Controlled Traffic

We first use controlled traffic to evaluate QAir's performance. We use Iperfto generate TCP/UDP traffic. The traffic is mixed with n background flows and 1 $ping$ flow. $ping$ acts as the delay sensitive traffic and reports the round trip time. We vary n to generate different extent of contention. In the results presented below, we focus on the case where there is only one delay sensitive flow ($ping$), as the number of $ping$ flows does not significantly impact the overall throughput and per-frame latency according to both our intuition and actual experiment results.

For each run, the experiment was repeated multiple times and using average to get stable results. We use two performance metrics: *the aggregated throughput* and *average RTT* measured by $ping$ tool. We also conduct extra experiments where we replace the $ping$ traffic with other delay-sensitive flows whose traffic demands range from 1Mbps to



(a) Aggregated Throughput.



(b) Average RTT to represent the packet level latency.

Fig. 8: The performance of QAir under UDP traffic.

7Mbps. As soon as one flow's traffic demand is smaller than $\frac{ChannelCapacity}{\#equivalent\ background\ flows}$ ($\approx \frac{70Mbps}{10} = 7Mbps$ in our testbed), this flow will not be regulated by our *Traffic Regulation* part (§ IV-C), thus the per-frame latency show the same results with $ping$ traffic. The results are ommited from this part because of the space limitation.

UDP: In the first scenario, all the background traffic are 100Mbps UDP flows, and we varied n, the number of such UDP flows. The results are shown in Figure 8. From Figure 8b we can see QAir can greatly reduce the RTT (packet level latency) compared to the baseline. When $n \leq 4$, RTT is reduced by 50% to 75%. The larger n and the contention, the better QAir works. On the other hand, Figure 8a shows that the QAir achieves similar throughput performance as the baseline.

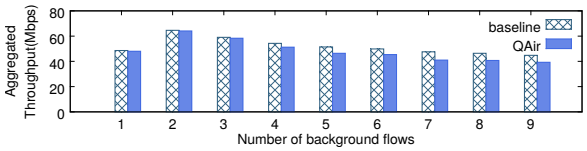
TCP: In the second scenario, the background traffic are n persistent TCP flows, which continues to send TCP packets according to what TCP congestion control allows. Note that a receiver also contends for the channel because it needs to send ACKs, thus the total number of contenders are actually between n and $2 \times n$. The results are shown in Figure 9. From Figure 9b we can see QAir can greatly reduce the RTT (packet level latency) compared to the baseline. When $n \leq 5$, RTT is reduced by 50% to 60%. The larger n and the contention, the better QAir works. Figure 9a shows that the QAir has only a minor throughput scarifies (at most 15%) compared to the baseline.

Above results show that QAir 's congestion control for the virtual queue in the air can complement TCP's congestion control at the transport layer.

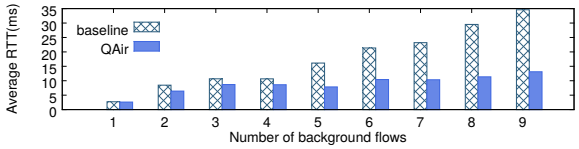
C. Performance of Real Applications

The evaluation using Iperf and $ping$ flows shows that QAir can greatly reduce the RTT of the wireless hop. We now use live streaming as an example to evaluate how QAir can help improve the performance of a real world application.

The settings are the same to those in §VI-B except that the delay-sensitive flow $ping$ is replaced with a live streaming mobile app. As shown in Figure 10, we broadcast a real-time



(a) Aggregated Throughput.



(b) Average RTT to represent the packet level latency.

Fig. 9: The performance of QAir under TCP traffic.

TABLE III: QoE of live streaming with/without QAir.

	#Buffering events	Minimum e2e Delay (s)	Maximum e2e Delay (s)
QAir disable	7	2.15	9.76
QAir enable	0	2.15	2.23

clock from one phone to another. We measure two important QoE (quality of experience) metrics for live streaming [30], [31]: *number of buffering events* and *end to end (e2e) delay*. e2e delay means how much time the "viewer" lags behind the "broadcaster", and buffering event means the streaming is paused to get more data from the network.

The number of buffering events can be easily counted from the app. By putting the "viewer" and "broadcaster" side by side, we can get the e2e delay from the clock difference on the two phones. The experiment was done for 100 seconds for both baseline and QAir. We summarize the results in Table III, which shows QAir can greatly improve the QoE of live streaming: free of buffering (*v.s.* 7 times in baseline) and short time delay between "broadcaster" and "viewer" (the maximum delay is reduced from 9.76s to 2.23s). Note that although the e2e delay consists of various parts (from the broadcaster app, to servers and CDNs *etc.*) [31], QAir has a very stable e2e delay ranging from 2.15s to 2.23s, which significantly helps the QoE.

It is noteworthy that the scenario when 9 background flows may be uncommon in real world. However, it is common that there are dozens of wireless node in one same place, *e.g.*, the scenario we measured in Figure 2. Each wireless node may not have background traffic, but the larger number of nodes make wireless hop RTT comparable with 9 background flows ($26ms$ for median RTT in Figure 2 and $35ms$ for average RTT in Figure 9b).

VII. Discussions

Diverse frame size and rate: In practice, WiFi frames carry different size of payloads, and use different data rates depending on link quality. A nice property of QAir is that it is not sensitive to frame size and data rate. As the control target of QAir is latency, when frame size or data rate changes, QAir will converge to the same latency, although to a different number of contenders. The number of contenders may be large in extreme cases, for example, if every device transmits very small frames with very high data rate, however, in our tests we did not observe any noticeable impact from diverse frame size and data rate.

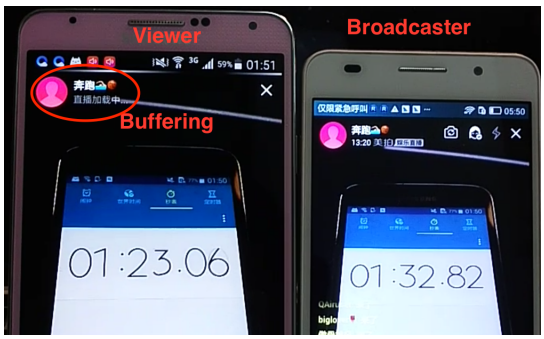


Fig. 10: Snapshot of live streaming of a real-time clock.

Hidden terminals: Hidden terminals cause additional frame losses and enlarged latency,therefore may falsely trigger QAir to regulate traffic. We suggest to mitigate hidden terminals using RTS/CTS. Specifically, when a device observes high frame loss rate which will not be reduced with lower data rate, the device should enable RTS/CTS exchange before every frame. According to our experience, similar hidden terminal mitigation mechanism has already been widely adopted in today's WiFi devices.

Legacy nodes: Since QAir nodes will regulate traffic upon observed high latency but legacy nodes will not, legacy nodes will gain unfair performance advantage over QAir nodes. For example, for best effort flows on QAir nodes, the obtained throughput will be smaller than expected after traffic regulation if some legacy nodes are transmitting heavily. Therefore, we suggest to deploy QAir to all the devices in an environment. For example, administrators can enforce the use of QAir in AP's admission control.

VIII. Conclusions

In this paper, we have designed and implemented QAir, a practical solution to reduce the latency in dense WiFi networks. QAir has a control loop of three steps: 1) measuring the contention level of the channel based on per-frame latency, 2) determining a proper control rate from the channel measurement and a control target, and 3) using the control rate to regulate the traffic from the MAC layer to the physical layer. QAir is thus able to transfer the distributed queue into host queue. Consequently, the delay-sensitive flows can bypass the entire queue and their latency can be greatly reduced. QAir works in a distributed manner with no centralized scheduler. Through real experiments, we demonstrate that QAir can sig-

nificantly reduce WiFi latency of delay-sensitive flows without sacrificing the network throughput.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedbacks. We are grateful to Yuanwei Lu, Bojie Li and Meng Meng for their helpful suggestions. We also thank Xinjian Yu for helping us set up the testbed. We thank Juexing Liao for her proofreading. This work has been supported by National Natural Science Foundations of China (NSFC) under Grant 61472210 & 61472214, the Sate Key Program of National Science of China under Grant No.61233007, the National Key Basic Research Program of China (973 program) under Grant No.2013CB329105, the Tsinghua National Laboratory for Information Science and Technology key projects, the Global Talent Recruitment (Youth) Program and the Cross-disciplinary Collaborative Teams Program for Science, Technology and Innovation, of Chinese Academy of Sciences-Network and system technologies for security monitoring and information interaction in smart grid.

REFERENCES

[1] Changhua Pei, Youjian Zhao, Guo Chen, Ruming Tang, Yuan Meng, Minghua Ma, Ken Ling, and Dan Pei. Wifi can be the weakest link of round trip network latency. In *INFOCOM, 2016 Proceedings IEEE*, 2016.

[2] S. Sundaresan, N. Feamster, R. Teixeira, N. Magharei, et al. Measuring and mitigating web performance bottlenecks in broadband access networks. In *ACM Internet Measurement Conference*, 2013.

[3] Martin Heusse, Paul Starzetz, Franck Rousseau, Gilles Berger-Sabbatel, and Andrzej Duda. Bandwidth allocation for diffserv based quality of service over 802.11. In *Global Telecommunications Conference, 2003. GLOBECOM'03. IEEE*, volume 2, pages 992–997. IEEE, 2003.

[4] IEEE Computer Society LAN MAN Standards Committee et al. Wireless lan medium access control (mac) and physical layer (phy) specifications, 1997.

[5] Stefan Mangold, Sunghyun Choi, Peter May, Ole Klein, Guido Hiertz, and Lothar Stibor. Ieee 802.11 e wireless lan for quality of service. In *Proc. European Wireless*, volume 2, pages 32–39, 2002.

[6] Daqing Gu and Jinyun Zhang. Qos enhancement in ieee 802. 11 wireless local area networks. *IEEE Communications Magazine*, 41(6):120–124, 2003.

[7] Sunghyun Choi, Javier Del Prado, Stefan Mangold, et al. Ieee 802.11 e contention-based channel access (edcf) performance evaluation. In *Communications, 2003. ICC'03. IEEE International Conference on*, volume 2, pages 1151–1156. IEEE, 2003.

[8] Stefan Mangold, Sunghyun Choi, Guido R Hiertz, Ole Klein, and Bernhard Walke. Analysis of ieee 802.11 e for qos support in wireless lans. *IEEE wireless communications*, 10(6):40–50, 2003.

[9] Zhifeng Tao and Shivendra Panwar. Throughput and delay analysis for the ieee 802.11 e enhanced distributed channel access. *IEEE Transactions on communications*, 54(4):596–603, 2006.

[10] Lamia Romdhani, Qiang Ni, and Thierry Turletti. Adaptive edcf: enhanced service differentiation for ieee 802.11 wireless ad-hoc networks. In *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, volume 2, pages 1373–1378. IEEE, 2003.

[11] Chadi M Assi, Anjali Agarwal, and Yi Liu. Enhanced per-flow admission control and qos provisioning in ieee 802.11 e wireless lans. *IEEE Transactions on Vehicular Technology*, 57(2):1077–1088, 2008.

[12] Inanc Inan, Feyza Keceli, and Ender Ayanoglu. Multimedia capacity analysis of the ieee 802.11 e contention-based infrastructure basic service set. In *IEEE GLOBECOM 2008-2008 IEEE Global Telecommunications Conference*, pages 1–6. IEEE, 2008.

[13] Inanc Inan, Feyza Keceli, and Ender Ayanoglu. A capacity analysis framework for the ieee 802.11 e contention-based infrastructure basic service set. *IEEE Transactions on Communications*, 57(11):3433–3445, 2009.

[14] Yang Xiao and Haizhon Li. Voice and video transmissions with global data parameter control for the ieee 802.11 e enhance distributed channel access. *IEEE Transactions on parallel and distributed systems*, 15(11):1041–1053, 2004.

[15] Patrick Verkaik, Yuvraj Agarwal, Rajesh Gupta, and Alex C. Snoeren. Softspeak: Making VoIP play well in existing 802.11 deployments. NSDI'09.

[16] Godfrey Tan and John V Guttag. Time-based fairness improves performance in multi-rate wlans. In *USENIX Annual Technical Conference, General Track*, pages 269–282, 2004.

[17] Vivek Shrivastava, Nabeel Ahmed, Shravan Rayanchu, Suman Banerjee, Srinivasan Keshav, Konstantina Papagiannaki, and Arunesh Mishra. Centaur: realizing the full potential of centralized wlans through a hybrid data path. In *Proceedings of the 15th annual international conference on Mobile computing and networking*, pages 297–308. ACM, 2009.

[18] Wenjie Zhou, Dong Li, Kannan Srinivasan, and Prasun Sinha. Domino: relative scheduling in enterprise wireless lans. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 381–392. ACM, 2013.

[19] Chao-Fang Shih, Yubing Jian, and Raghupathy Sivakumar. Look whos talking: A practical approach for achieving scheduled wifi in a single collision domain. In *ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2015.

[20] Giuseppe Bianchi and Ilenia Tinnirello. Kalman filter estimation of the number of competing terminals in an ieee 802.11 network. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 2, pages 844–852. IEEE, 2003.

[21] Luciano Bononi, Marco Conti, and Enrico Gregori. Runtime optimization of ieee 802.11 wireless lans performance. *IEEE Transactions on Parallel and Distributed Systems*, 15(1):66–80, 2004.

[22] Frederico Calì, Marco Conti, and Enrico Gregori. Dynamic tuning of the ieee 802.11 protocol to achieve a theoretical throughput limit. *IEEE/ACM Transactions on Networking (ToN)*, 8(6):785–799, 2000.

[23] Hui Ma, Xing Li, Hewu Li, Peiyun Zhang, Shixin Luo, and Cong Yuan. Dynamic optimization of ieee 802.11 csma/ca based on the number of competing stations. In *Communications, 2004 IEEE International Conference on*, volume 1, pages 191–195. IEEE, 2004.

[24] Tiantong You, Chi-Hsiang Yeh, and Hossam Hassanein. A new class of collision prevention mac protocols for wireless ad hoc networks. In *Communications, 2003. ICC'03. IEEE International Conference on*, volume 2, pages 1135–1140. IEEE, 2003.

[25] Martin Heusse, Franck Rousseau, Romaric Guillier, and Andrzej Duda. Idle sense: an optimal access method for high throughput and fairness in rate diverse wireless lans. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 121–132. ACM, 2005.

[26] Saverio Mascolo, Claudio Casetti, Mario Gerla, Medy Y Sanadidi, and Ren Wang. Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. In *MobiCom*, pages 287–297. ACM, 2001.

[27] Radhika Mittal, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, David Zats, et al. Timely: Rtt-based congestion control for the datacenter. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 537–550. ACM, 2015.

[28] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN systems*, 17(1):1–14, 1989.

[29] Yu-Chung Cheng, Mikhail Afanasyev, Patrick Verkaik, Péter Benkö, Jennifer Chiang, Alex C Snoeren, Stefan Savage, and Geoffrey M Voelker. *Automating cross-layer diagnosis of enterprise wireless networks*, volume 37. ACM, 2007.

[30] Matti Siekkinen, Enrico Masala, and Teemu Kämäräinen. A first look at quality of mobile live streaming experience: The case of periscope. In *Proceedings of the 2016 ACM on Internet Measurement Conference*, IMC '16, pages 477–483, New York, NY, USA, 2016. ACM.

[31] Bolun Wang, Xinyi Zhang, Gang Wang, Haitao Zheng, and Ben Y. Zhao. Anatomy of a personalized livestreaming system. In *Proceedings of the 2016 ACM on Internet Measurement Conference*, IMC '16, pages 485–498, New York, NY, USA, 2016. ACM.