TCP Download Performance in Dense WiFi Scenarios: Analysis and Solution

Mukulika Maity, Bhaskaran Raman, and Mythili Vutukuru

Abstract—How does a dense WiFi network perform, specifically for the common case of TCP download? While the empirical answer to this question is 'poor', analysis and experimentation in prior work has indicated that TCP clocks itself quite well, avoiding contention-driven WiFi overload in dense settings. This paper focuses on measurements from a real-life use of WiFi in a dense scenario: a classroom where several students use the network to download quizzes and instruction material. We find that the TCP download performance is poor, contrary to that suggested by prior work. Through careful analysis, we explain the complex interaction of various phenomena which leads to this poor performance. Specifically, we observe that a small amount of upload traffic generated when downloading data upsets the TCP clocking, and increases contention on the channel. Further, contention losses lead to a vicious cycle of poor interaction with autorate adaptation and TCP's timeout mechanism. To reduce channel contention and improve performance, we propose a modification to the AP scheduling policy to improve the performance of large TCP downloads. Our solution, WiFiRR, picks only a subset of clients to be served by the AP during any instant, and varies this set of "active" clients periodically in a round-robin fashion over all clients to ensure that no client starves. We have done extensive evaluation of WiFiRR in simulation and in real settings. By reducing the number of contending nodes at any point of time, WiFiRR improves the download time of large TCP flows upto $3.5\times$ of our classroom scenario. We also compare WiFiRR with state-of-the-art prior work WiFox, WiFiRR improves download time by $2.25\times$ over WiFox.

Index Terms—Dense WiFi network, TCP download performance, channel contention, real setting, scheduler

1 Introduction

The omni-presence of WiFi needs no justification. While WiFi standards have improved significantly in terms of raw bit-rate, whether this has translated to corresponding improvements in application throughput is unclear. We are specifically interested in dense user scenarios, such as conferences, sports stadiums, and large classrooms, with the latter two being especially nascent with respect to WiFi usage. How does a dense WiFi network perform, specifically for the common case of TCP downloads? This is the focus of our work.

Prior work has shown, both analytically [1], [2] and experimentally [3], [4], that TCP download performance does not degrade with increasing number of users in a WLAN. These results are based on the performance of long running TCP flows in controlled environments, using homogeneous well-tested clients and artificial user traffic. These studies have reported good TCP download performance even with over a hundred clients [3].

In contrast, this paper presents a measurement study of TCP performance "in the wild" over a dense WiFi network, with real users running real applications over a variety of client devices. We conduct several measurements in several WiFi-enabled classrooms, where students download online quiz questions and/or instruction material. Our results show that, in contrast to prior work, TCP performance degrades significantly in a dense usage scenario, even with 20-30 clients per access point. (We focus on a single WiFi BSS, and do not address scaling issues across multiple interfering BSSs.)

We have analyzed why our results differ from the TCP download scenarios in prior research. With long running TCP downloads, the only traffic on the network is TCP data packets in the downlink and ACKs in the uplink. In such cases, the number of contenting nodes on the channel is usually quite low, because the AP alone transmits TCP data, and only the clients that most recently received a data packet are likely to contend for the channel to send an TCP ACK. In contrast, in our real-life measurements, we found significantly higher channel contention due to "chattiness" of real applications that create a small but noticeable amount of extra upload traffic besides TCP ACKs.

For example, in one of our classroom scenario, a student logs in to the class webpage, authenticates herself, locates a file to download on a webpage (that has several smaller web objects in addition to the main object of interest), using a browser that opens several parallel TCP connections to download the content. In addition, users also have a low volume of background traffic automatically generated by e-mail clients and such. Somewhat surprisingly, this small amount of extra traffic in the upload direction significantly increases the contention on the channel (as the number of active clients is now close to the total number of users), resulting in collisions due to the CSMA MAC protocol's channel arbitration mechanism. As a result, we found that TCP performance degraded severely, and students often took more than $8\times$ the amount of time to download the files needed for an in-class quiz, as compared to a universe where TCP scaled perfectly with increasing user density.

• *The authors are with the Department of CSE, IIT Bombay, Mumbai, Maharashtra, India. E-mail: {mukulika, br, mythili}@cse.iitb.ac.in.*

Manuscript received 24 Aug. 2015; revised 5 Nov. 2015; accepted 25 Feb. 2016. Date of publication 10 Mar. 2016; date of current version 1 Dec. 2016. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TMC.2016.2540632

We find that the contention on the wireless channel and the resulting collision losses also have an undesirable effect on several other protocols in the system. For example, we observed that WiFi clients picked lower bit rates during (and for a short period of time after) contention, because most rate adaptation algorithms confuse collisions for channel losses. This lowering of rate increases the time taken for subsequent transmissions, further increasing contention, leading to a vicious cycle. Further, we observed poor interaction between channel contention and TCP's timeout mechanism. We found that the RTT of TCP flows was highly variable due to contention losses, confusing the TCP timeout algorithm, leading to spurious retransmissions. Note that while prior work [5], [6], [7], [8], [9], [10] has also observed some subset of these problems, our analysis has focused on thoroughly identifying almost *all* factors that contribute to poor TCP download performance in dense scenarios, and understanding their complex interplay.

We also observed that in practice, several device drivers become unresponsive when operating under high contention losses, and need a driver reset to function even after the contention has subsided. All of these real-life effects further exacerbate the TCP performance issues in a dense setting. Note that we have verified and eliminated other factors (AP buffer mismanagement, wired network/server overload, external interference) as possible causes for the poor performance.

Having identified excessive channel contention as the root cause behind the performance issues, we propose a solution, WiFiRR, to improve the performance of large TCP downloads in dense WiFi scenarios. WiFiRR works as a scheduler at the packet queue of an access point. WiFiRR identifies a subset of clients as "active" during every instant of time (up to five clients in our implementation), and the AP serves downlink packets only to these clients. This results in the other clients going quiet during this period, leading to lower contention and improved performance. This set of active clients is varied periodically (every 1.6 s in our case) to cover all clients in a round-robin fashion, hence the 'RR' in the name WiFiRR. Note that while clients may temporarily be deprived of service for short durations, they will eventually see improved performance over large TCP downloads. We evaluate our solution in simulation and in real experiments. In simulation, we emulate the classroom browsing behaviour by creating our own ns-3 application modules *ChattyClient* (client) and *WebServer* (server). Here, WiFiRR improves TCP download time by $1.6\times$ over the base case. Next, we evaluate WiFiRR in real settings. We set up a testbed of 15 clients connected to one 802.11n access point. We emulate the browsing behaviour observed in classroom by using a browser replacement tool i.e., Epload [11]. Here, WiFiRR improves TCP download time by $2.4\times$ over the base case.

We soon realized that our AP side solution of suppressing downlink traffic in the view reducing uplink contention might not be effective if the uplink traffic continues without downlink traffic (for ex. TCP SYN retries). Thus to reduce the client side chattiness even more we seek client side solution. The new HTTP protocol, SPDY [12] helps in reducing the client side chattiness. SPDY opens a single TCP connection for downloading multiple web objects. This helps in reducing the contention. We have evaluated SPDY both in simulation and in real settings. We have found WiFiRR provides most improvement when tried with SPDY compared to HTTP. In simulation, HTTP+WiFiRR provides improvement of $1.6\times$, SPDY+WiFiRR provides improvement of $2.9\times$. In real settings, HTTP+WiFiRR provides improvement of $2.4\times$, SPDY +WiFiRR provides improvement of $2.7\times$.

We examine scalability of WiFiRR by evaluating it for different number of clients. We vary the total number of clients from 10 to 50. WiFiRR provides maximum improvement of $3.5\times$ when the network size is 40. The improvement of WiFiRR does not degrade with increasing number of clients.

We also compare our solution to another solution WiFox [5] that seeks to improve TCP download performance in dense scenarios by prioritizing AP. WiFiRR improves download time by $2.25\times$ over WiFox. To reduce contention, WiFox merely prioritizes the AP over clients while we address contention more directly by maintaining only a few 'active' clients while throttling all others. We have also evaluated WiFiRR for different interactive applications like skype, ssh etc. We realize that WiFiRR does not give significant performance improvement for these traffic types but nor does it degrade performance for these traffic types.

Our contributions can be summarized as follows: (a) a real-life measurement study of TCP download performance and its careful analysis, which identifies the factors that contribute (and eliminates the factors that do not) to poor performance in dense scenarios, and (b) a solution approach WiFiRR that improves the download time of large TCP flows by reducing channel contention, and (c) extensive evaluation of WiFiRR in simulations and in real settings.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 describes our measurement study in real classrooms, and Section 4 describes some controlled experiments and simulations we conducted to understand the measurement results in the classrooms. Section 5 describes the design of our solution WiFiRR that improves performance by addressing the problems we found. Section 6 presents evaluation of the solution in simulation and in real settings. Finally, Section 7 concludes the paper.

2 Related Work

Starting with Bianchi's seminal work [13], several researchers have analytically shown that the performance of 802.11 CSMA/CA degrades with increase in offered load, due to increased contention on the wireless channel. This analysis assumes saturated traffic, i.e., all stations are always backlogged and contend for the channel. [14] further generalizes the result, and shows that collision probability increases with increasing number of stations. However, subsequent research [1], [2] has considered a more specific problem of TCP downloads over 802.11. In this case, the analysis shows that the number of contending stations is much lower than the total number of stations due to the TCP data/ack clocking mechanism. When several downlink flows go through an AP, and the AP sends a data packet to a client at a certain instant, the client that received this data packet alone will generate a TCP ACK, and contend with the AP for the channel. All the other clients will not actively contend for the channel at this instant, until data packets arrive for them from the AP. This data/ack clocking mechanism of TCP

flows ensures that the contention on the channel and collision probability stay low, with the result that the system throughput does not degrade much with increasing number of clients.

The analysis results of the scaling of TCP downloads have also been backed up by experimental studies [3], [4]. These papers show that the TCP's data/ack clocking mechanism allows TCP downloads to scale to over hundred clients without any significant degradation of aggregate system throughput. However, the experiments in these papers consider only long running TCP flows and emulated user traffic on testbeds of homogeneous nodes. In contrast, our measurement study conducted with several tens of users trying to download files using TCP shows that TCP download does not scale as well in the context of real user traffic.

Several researchers have reported some subset of the problems we have encountered in our measurement study, and suggested several techniques to address these problems. Prior work [5], [6], [7] has considered the problem of asymmetry between uplink and downlink traffic in WLANs. When a large number of users are downloading traffic over the WLAN, most traffic is downlink. However, the AP that delivers all the downlink traffic has to contend for the channel with the other clients, resulting in an unfair allocation to the downlink traffic. To solve this problem of asymmetry, these papers propose several MAC-layer enhancements to prioritize the AP's channel access. For example, WiFox [5] prioritizes AP's channel access over the clients dynamically depending on the load in the network. The AP accesses medium with high priority when AP's transmission queue size is high and accesses with default priority otherwise. As a result, WiFox claims to give 400-700 percent increase in throughput and 30-40 percent improvement in average response time. Our work differs from WiFox and other related work in that we identify and address several factors (besides asymmetry between uplink and downlink) that contribute to poor TCP download performance in dense scenarios. Specifically compared to WiFox, our work differs in that while WiFox merely prioritizes the AP over clients, we address contention more directly by throttling all except a few 'active' clients.

Other research [8] has observed the effect of channel contention on the RTT of TCP flows through a WLAN. The authors show that highly variable RTTs due to contention lead to incorrect estimation of TCP retransmission timeout, and hence lead to spurious retransmissions. The authors propose prioritization of TCP ACKs as a solution. Researchers have also observed the impact of channel contention on bit rate adaptation [9], [10] in dense deployments, and proposed solutions to prevent lowering of bit rate unnecessarily in response to collisions. While each of the above papers measure and analyze a subset of problems that arise in a dense WiFi network, none of them have reported all of the problems or their complex interplay we find in our measurements.

SPDY [12] is an enhancement to HTTP/1.1 protocol. SPDY pronounced as "SPeeDY" is an application layer protocol developed primarily at Google to speed up the web. The basic features of SPDY are: 1) TCP stream multiplexing, 2) HTTP header compression, 3) Request prioritization and, 4) Server push. 1) TCP stream multiplexing: SPDY opens a single TCP connection for multiple web objects. It multiplexes requests for multiple web objects on a single connection. This increases the efficiency of TCP as fewer network connections are made. 2) HTTP header compression: SPDY compresses request and response headers. So, the total number of transmitted packets and bytes reduces. 3) Request prioritization: SPDY assigns a priority to each request thus higher priority requests are served before the non-critical requests. 4) Server push: SPDY allows the server to push the content to the client even before it requests them. This enables effective usage of the bandwidth. SPDY is complementary to our work. Any application layer protocol modifications or client side solutions are complementary to our work. In our evaluation, we show that the use of SPDY in fact magnifies WiFiRR's gain further.

A new IEEE standard IEEE 802.11ah [15] is coming up which utilizes 1 GHz license-exempt bands. It basically supports the concept of the Internet of Things (IoT). Here large number of stations/sensors will co-operate to share the network channel, thus the contention on the network will be high. To reduce the contention/collision, the stations are divided into several groups. The network channel access is divided into two tiers: inter-group and intra-group. Two such grouping algorithms exist following traditional IEEE 802.11 standard, they are: Token-coordinated random access MAC (TMAC) [16] and Group-based MAC (GMAC) [17]. In TMAC protocol, the coordinating AP assigns a non-overlapping interval (slot) to each group. Contention happens only within the group. In case of GMAC, leaders from each group are selected. Contention happens only within the leaders. Then the winning leader specifies the schedule of the transmissions from the group and stations transmit accordingly. [18] improves the above TMAC and GMAC protocols in the context of Smart Metering Network (SMN) for the new IEEE 802.11ah standard and proposes enhanced version of TMAC and GMAC protocols. They are: TDMA-DCF and DCF-TDMA respectively. These two protocols take care of hidden terminal problem, static nodes etc. The authors have done numerical analysis and have proposed optimal group size for the above two protocols.

In case of IEEE 802.11ah, the scale is really high. For e.g., IEEE 802.11ah TG needs to support 6,000 smart meters in smart grid use case [19]. However, we face performance issues even with 20-30 devices per AP. Our idea is also similar to these protocols, as we too maintain only a subset of clients to be active in channel contention. But for our case, group division is done implicitly. We do not require client-side support or MAC protocol modification. WiFiRR works for traditional IEEE 802.11 standard.

To summarize, our work improves over prior work on improving TCP performance in real-life dense scenarios by analyzing the problem more thoroughly, and identifying interplay of almost all the factors that contribute to poor performance.

3 Measurements in Live Classrooms

As discussed in the previous section, prior work has shown analytically and experimentally that TCP download performance scales well. So, will the good performance observed in these analysis and lab experiments carry over to real life?

TABLE 1
Measurement Dataset

Parameter	Classroom-1	Classroom-2	Classroom-3
Number of WiFi clients	94	200	43
Number of Access Points	3	3	1
Activity	Online quiz	Online quiz	Instructional material (video) download
Clients	Mostly laptops and few phones/tablets	Mobile phones/tablets	Mostly laptops and few phones/tablets

To understand this, we collected wireless and TCP measurements for several courses running at our Institute in live classrooms. WiFi was used in the classrooms for various classroom activities i.e., online quiz, download of instructional materials etc.

Here we present three such sample scenarios in three different physical classrooms, and name them as classroom-1, classroom-2 and classroom-3. Table 1 summarizes our measurement data set showing the number of WiFi clients, number of access points, classroom activities and type of clients for the three classrooms.

3.1 Classroom-1

We first describe our classroom-1 measurements.

3.1.1 Measurement Setup

We describe our data collection method for measurements in a real classroom. In a course with 124 registered students, taught by one of the authors, a subset of lectures involved downloads of supplementary instruction material by students, and some involved graded quizzes. The students used individual laptops and tablets, and some desktops as well, for these activities. Our setup consisted of students connecting to a web server that hosts instruction or quiz content. A small fraction of students used wired access. We had three enterprise-grade WiFi APs, setup in the three non-overlapping 802.11g channels 1, 6, and 11. All the relevant entities were on the same extended LAN.

The activity was as follows. The students browsed to the content server, authenticated themselves, and downloaded a variety of content (video lectures, references, quizzes) over the wireless channel as instructed. We instrumented the web server to log the per-request service time. In addition, we also collected network traces from two vantage points: (i) The WiFi AP was instrumented to collect per-frame MAC layer statistics. Our code had access to hardware registers in the WiFi NIC, that let us determine the fraction of airtime that was spent in transmissions, receptions, and in idle listening at a very fine granularity of 250ns. (ii) A sniffer running tcpdump was connected via an Ethernet hub to the content server to collect TCP and HTTP logs.

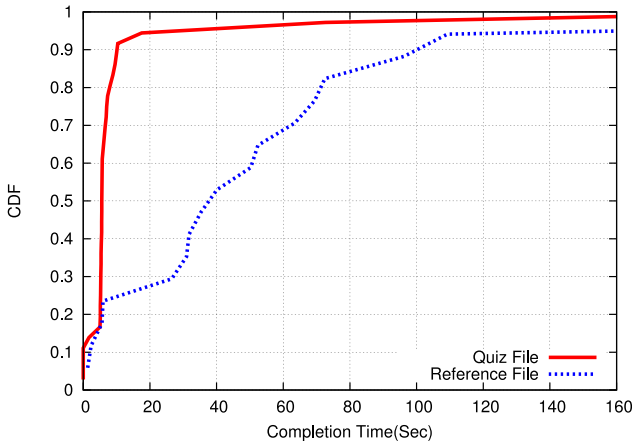


Fig. 1. CDF of the time taken to download the quiz and references files.

Prior to our measurements, we ensured that the WiFi AP, the web server, and the wired backhaul from the AP to the web server were not loaded. That is, the performance seen by the clients was constrained by the wireless network bottleneck. External WiFi interference was minimal.

From all our measurements, we choose one representative dataset to present results from: a quiz conducted in class. In the quiz, 94 students, spread roughly equally over 3 APs, downloaded a quiz question paper of size ≃200 KB. A subset of 24 students also downloaded the optional reference material file of size ≃4 MB. We pick one of the three APs to present results from; the results at the others were similar. This AP in question served 32 students: all 32 students downloaded the quiz file, and 17 students downloaded the reference material.

3.1.2 Results

First, we present the most important performance metric—the completion time, since this delay determines how users perceive the quality of the network. The completion time is measured as the time from the issue of HTTP GET request for the particular file to the last packet of the download received by the user. Fig. 1 shows the CDF of the completion times for all the clients, for both the quiz and reference files. To put these numbers in perspective, let us calculate the expected download time. First, note that our classroom was such that even the farthest client could comfortably operate at the highest 54 Mbps bit rate of 802.11g, when operating in isolation (we verified this during AP placement). This physical layer bit rate translates to about 24 Mbps of TCP-layer throughput, after accounting for link-layer overheads and the overheads of TCP ACKs. If we go by prior work that claims that TCP download throughput scales perfectly with the number of clients, each client should have gotten a TCP throughput of 24 Mbps/32=0.75 Mbps. Assuming all clients downloaded both the quiz and reference file, which we overestimate as 5 MB worth of content per client, the expected download time still works out to only about 5 MB/0.75 Mbps=54 s. In contrast, the worst case completion times in Fig. 1 was 229 s for the quiz file, and 478 s for the reference file![1]

1. This of course created logistical problems; the instructor had to give time extensions to those students who experienced delay in downloading!

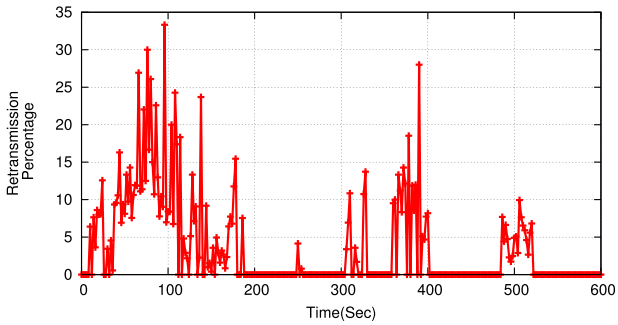


Fig. 2. The average TCP retransmission rate across all clients versus time.

Next, we investigate why the completion time was so bad. Upon looking at the TCP time-sequence graphs, we found that some clients suffered severe TCP segment losses, and often timed out several times during the course of the measurement. Fig. 2 shows the average TCP retransmission rate (averaged across all clients every 2 s) as a function of time; most of these retransmissions were due to a TCP timeout. Fig. 3 shows the TCP time sequence graph of a client that experienced multiple TCP timeouts.

To understand why the application-layer performance was so bad, we analyze the logs collected at the AP to understand the MAC-layer performance in the network. Using our custom instrumentation of the AP driver, we determined what fraction of the airtime was reported as "busy" at the AP. This air occupancy percentage is shown in Fig. 4 for the duration of the quiz download. Also shown is the aggregate download throughput of the AP during this time. Both measures are shown as two-second averages. We see from the figure that there are periods where the channel is busy, and there are also periods where the channel is idle for large fractions of time. This behavior fits well with our earlier observation of TCP timeouts. We also note that, irrespective of the channel busy percentage, the aggregate throughput is poor most of the time.

We further analyze the AP's logs to determine what caused the AP to deliver such low throughput, even when the channel was busy. We verified that the signal strength at all clients was good enough to support high bit rates. The other possibility is that of collisions, due to multiple clients picking the same backoff counter and transmitting in the same slot during CSMA MAC's channel arbitration mechanism. Collisions are notoriously hard to detect using packet

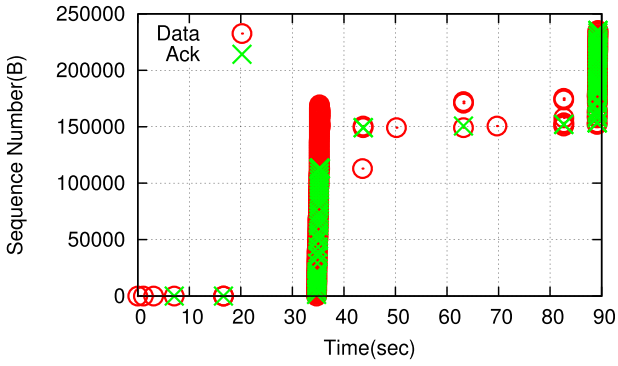


Fig. 3. TCP time sequence diagram of a client that experienced multiple timeouts.

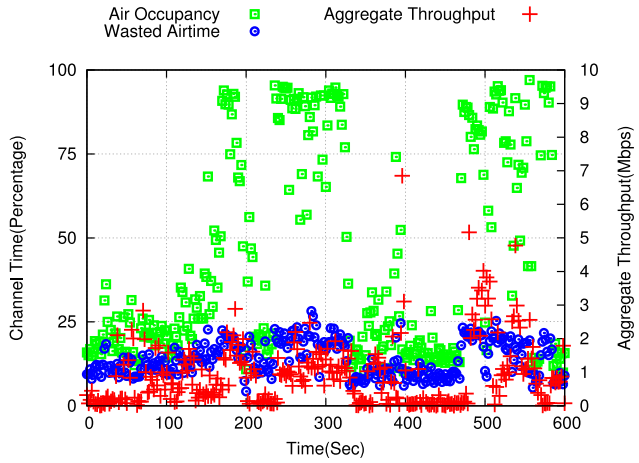
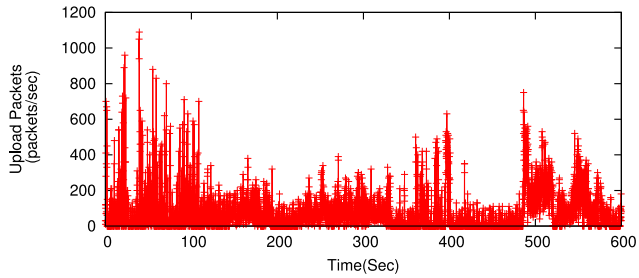


Fig. 4. Aggregate throughput, air occupancy, and wasted airtime at the AP.

logs, because collisions often result in a synchronization error at the physical layer, thereby leaving no trace in any kind of packet tracing mechanism. So, we use hardware registers exported to the device driver to determine the amount of "wasted" airtime at the AP, defined as the amount of time spent in one of the following activities: (i) transmitting packets that failed to elicit a link-layer ACK, (ii) receiving packets which could not be successfully decoded (either due to synchronization error at the physical layer, or a CRC error after synchronizing with the transmission). Fig. 4 also shows this wasted airtime percentage at the AP as a function of time. This figure shows that around 10-20 percent of the AP's airtime is often wasted, possibly due to collisions on the channel.

We next investigate why the contention and collision rate on the channel was so high. Prior work, as discussed in Section 2, shows that if the only traffic on the channel is TCP data and ACKs, the contention on the channel should be very low. However, in our measurement, we found that there was a small amount of extra upload traffic besides the TCP ACKs. Figs 5a and 5b show the rate of upload traffic in packets/sec and in kbps respectively. These metrics are shown as averages over 100 ms intervals; this indicates the burstiness of the upload traffic. This traffic consists of GET requests for various embedded objects on the course webpage, traffic generated in navigating the authentication page, TCP handshake packets for the multiple connections the browser opens, and some small amount of extra background traffic likely generated by email clients, Dropbox, and other such applications. Note that the amount of upload traffic is very low, averaging at about 8kbps in aggregate across all clients at the AP. However, it appears that this traffic was enough to increase the contention on the channel, and cause collision losses.

To investigate whether the upload traffic is the major reason for such poor performance we set up an controlled experiment. We connected 30 clients to an access point. The clients downloaded a 5 MB file through command line using WGET command as opposed to using a browser to open the classroom webpage, authenticating and then download. We shut down all the background traffic like Dropbox, email clients etc. The completion times were matching with the theoretical expectation. Given that



(a) Upload traffic (pkts/s) (b) Upload traffic (kbps)

Fig. 5. Upload traffic pkts/s & kbps generated by clients during the quiz.

WGET completion times are way lesser than the classroom scenario this points to the fact that upload traffic is one of major factors for poor performance.

The contention on the channel due to a large number of active clients is further exacerbated by the interaction with the bit rate adaptation. It is well known in prior work that most rate adaptation algorithms mistake collision losses for poor signal on the channel, and lower the bit rate in the hope of increasing the probability of packet delivery. However, transmissions at lower bit rate take up more airtime, further increasing the contention on the channel. Fig. 6 shows a CDF of the time-averaged bit rates of the clients during the quiz. We see that most clients were operating at a very low average rate, suggesting that the rate adaptation algorithms were using lower rates upon observing losses.

In addition to the metrics reported above, the overall experience of the students in using the WiFi network was very bad. When the students were simultaneously downloading large files and stressing the wireless networks, students often complained that their WiFi was not responding. We found many instances where a driver reset was needed to get the WiFi interface to work, even after the contention had subsided. We conjecture these to be possible device driver bugs that were triggered under the high loss rate situations we encountered in class. It is likely that such situations are not well tested in client driver code.

Below we present the measurements from two such scenarios. Since the performance in these cases were similar to that in classroom-1, we present only the MAC layer performance in each case.

3.2 Classroom-2

Next, we describe our classroom-2 measurements.

3.2.1 Measurement Setup

In this classroom almost 200 students were present. We installed three access points in three non overlapping channels of 2.4 Ghz ISM band. The WiFi network was used in classroom for online quiz activity using an Android quiz application on students' tablets. The activity is as follows: the students authenticated themselves to a server via the app. When the quiz is published by the teacher, the app downloaded the quiz file and enabled students to answer the questions. After that, responses are submitted to the server.

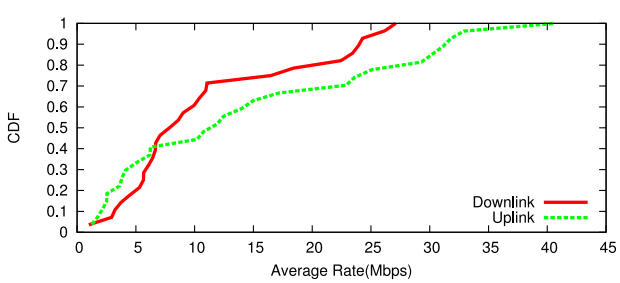


Fig. 6. CDF of the time-averaged bit rates of clients in the uplink and downlink directions.

3.2.2 Results

The students' experience of using WiFi network was really bad. The download time of quiz file was high. For some students, the download was stuck, they had to restart the app, to download afresh. They faced the same scenario during submitting answers as well. Out of 200 students, only about 120 students could actually submit their responses. This resulted in the instructor fall back on paper-printed quiz forms.

Out of the three access points we present result from one access point (the others were similar) which served 80 clients. Given that quiz file itself was small, we did not expect such poor performance. Fig. 7a shows MAC layer performance over a sample duration of 300 sec (similar graph as of Fig. 4 for classroom-1). Here we see although air is always busy, the aggregate throughput of the network is very low (less than 1 Mbps). To get an idea of the contention/collision on the channel, we determine the amount of wasted airtime. The wasted airtime is about 20-40 percent, which is even higher than in the case of classroom-1.

3.3 Classroom-3

Finally, we describe our classroom-3 measurements.

3.3.1 Measurement Setup

In this classroom setting, 43 students were associated to one AP. The WiFi network was used in classroom for downloading online video instructional material. The students used WiFi-enabled individual laptops and tablets for downloading video content from web server. The classroom activity is as follows: the students browsed to the content server, authenticated themselves and viewed embedded video content (with javascript controlled video markers) over the network.

3.3.2 Results

The empirical observation was that the network was noticeably slow when all the students were using the network. Fig. 7b shows MAC layer performance over a sample

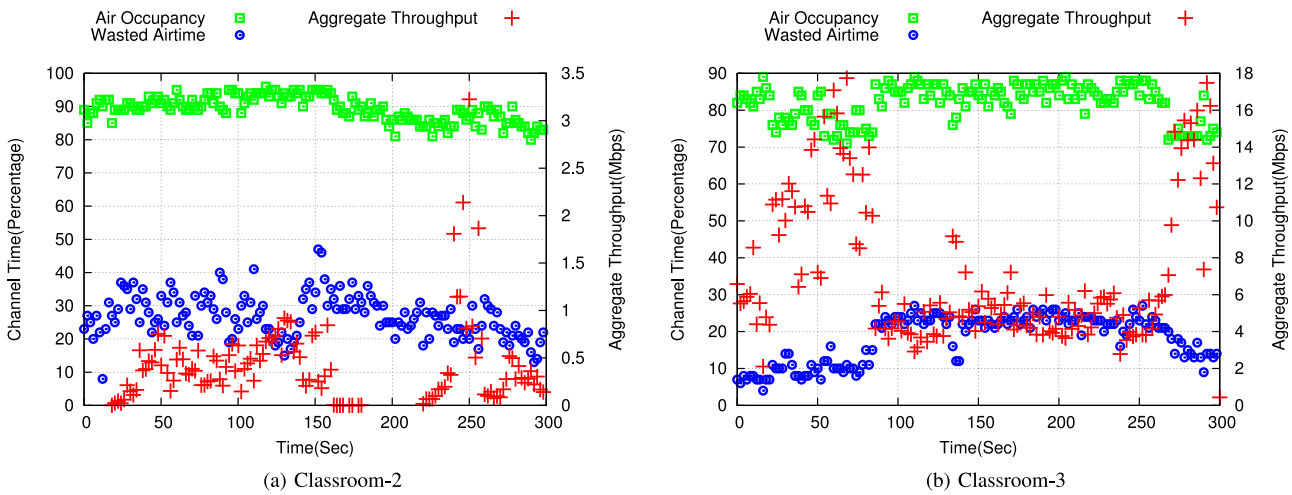


Fig. 7. Aggregate throughput, air occupancy, and wasted airtime at the AP for Classroom-2 and Classroom-3.

duration of 300 sec for this classroom. Here we see an interesting pattern, when the wasted airtime is low (10 percent) then the aggregate throughput is high (12-14 Mbps) as soon as the wasted airtime increases (20-30 percent), the aggregate throughput reduces (4-5 Mbps). This might be pointing to the fact that as the collision on the channel (thus the wasted airtime) increases the aggregate throughput reduces. Are collisions due to the CSMA MAC protocol mechanisms alone enough to explain the high losses we saw in our measurements? Or were there any other factors at work? The limited control to vary parameters and monitor performance in a live measurement makes it difficult to answer some questions, which we seek to address with a combination of simulations and controlled experiments in the next section.

4 Simulations and Controlled Experiments

In this section, we describe several results from simulations and controlled experiments conducted to better understand the classroom WiFi measurements.

4.1 Simulations

As described earlier, accurately measuring collision losses in an experiment is a hard problem. Even using multiple wireless sniffers on the air cannot guarantee that we can identify the error rate due to collisions, because the sniffers themselves may fail to decode most collisions. Therefore, we resort to simulations, where we can instrument the simulator to identify collisions.

We use the ns-3 network simulator to perform simulations. Our simulation model consists of 30 802.11g WiFi clients connected to an AP. To simulate the traffic seen in the classroom, each client emulates the browsing behaviour observed in classroom-1 and simultaneously downloads a 5 MB file from a server. We use *BulkSendApplication* module (predefined in ns-3) for downloading 5 MB file. ns-3 does not support any type of browsing module. So, we created two application modules in ns-3 for client and server. We have named them as *ChattyClient* for client module and *WebServer* for server module. In the *ChattyClient* module, one can specify number of parallel connections to be opened, number of web objects and their sizes. In the *WebServer* module one can specify response sizes. To emulate the browsing behaviour observed in classroom, each client opens two TCP connections and sends request for 20 objects. The request sizes are within 300-700 Bytes and response sizes are within 500-900 Bytes. The average upload rate is 10 kbps: close to what we observed in our classroom measurements. The clients are placed close enough to the AP to eliminate the possibility of any channel losses, so that a packet loss occurs only if two clients pick the same backoff value and collide during the CSMA MAC operation. We have used the Minstrel rate adaptation algorithm, and an AP queue size of 512 packets, as used in the real AP. Table 2 summarizes the parameters in our simulations.

TABLE 2
Simulation Setup

Parameter	Value
WiFi Protocol	802.11 g
Rate adaptation algorithm	Minstrel
AP queue size	512 packets
Number of clients	30
Download size	5 MB
Upload traffic	Browser behaviour

Our simulation resulted in a download time between 297 and 443 seconds for the clients, which roughly matches the download performance seen during the classroom quiz. During the download, the collision rate on the channel (defined as the fraction of airtime on the channel that was wasted in collision) shown in Fig. 12[2] was between 15 and 20 percent, proving that the poor TCP performance was primarily due to collision losses on the channel.

4.2 Controlled Experiments

Next, we perform several small-scale controlled experiments in the lab to better understand the impact of collisions and contention on the channel. We first seek to understand the impact of collision losses on the rate adaptation algorithm. We set up an experiment where a WiFi client

2. Fig. 12 is part of WiFiRR's evaluation and appears in sequence later.

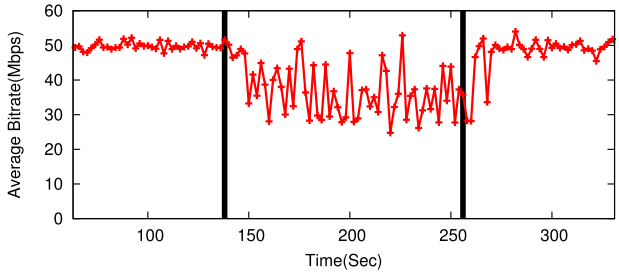


Fig. 8. The bit rates chosen by a WiFi client in the presence of contention.

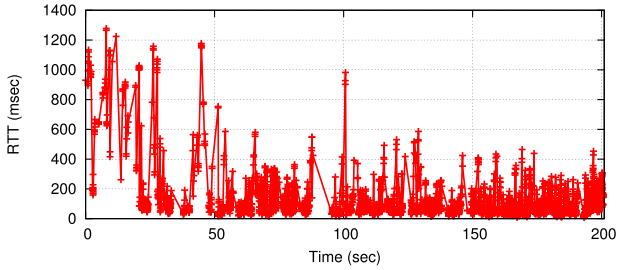


Fig. 9. Highly variable RTTs of a TCP flow caused by high channel contention.

(a Linux laptop with a "Qualcomm Atheros QCA9565 / AR9565 Wireless Network Adapter (rev 01)") is downloading a large file via the AP. After 2 minutes, we introduce 14 other WiFi clients on to the channel. These clients are Microtik single-board computers, that generate TCP upload traffic at the rate of 100 kbps. After a further 2-minute duration of high channel contention between the 15 WiFi clients, we turn off the Microtik boards, and let the laptop traffic run for some more time. Fig. 8 shows the time-averaged bit rate (averaged over 2 sec) of the laptop in the upload direction for the duration of this experiment. We observe from the figure that the rate adaptation algorithm lowers its bit rate due to collision losses, as expected. Further, we note that the rate adaptation algorithm takes around 12 seconds (after the Microtik boards are turned off) to recover from the effects of channel contention, which is approximately the timescale at which popular bit rate adaptation algorithms adapt rates. Similar results were observed with three other laptops running three different device drivers as well. This recovery time of the bit rate adaptation algorithm has a significance in explaining our measurement data of the previous section. The bursty upload traffic seen in our experiment has several periods of quiet between bursts of high upload activity. However, due to the relatively long recovery time of the rate adaptation algorithm, the effects of contention (i.e., the lowering of bit rates) persist even in the periods between upload bursts, magnifying the effect of the small amount of upload traffic.

Next, we identify the impact of collision losses on TCP performance. We setup an experiment with 21 WiFi clients (seven laptops and 14 Microtik single-board computers) connected to an AP. The clients download a 9 MB file from a server connected to the AP. The clients run a browser emulation script that generates around 50 GET requests to download several embedded objects in a sample webpage. In addition, the clients also generate a bursty upload TCP traffic at an average rate of 200 kbps, to simulate other background application traffic. This mix of upload and download traffic created enough contention on the channel to slow down the TCP downloads, and we observed several of the effects noticed in the classroom measurements. A wireless sniffer over the channel reported that 22 percent of frames decoded were marked as link-layer retries. While this is not a true indication of the collision rate on the channel (as the sniffer may have missed capturing several frames due to synchronization errors caused by collisions etc.), it gives us enough indication that the loss rate due to collisions is substantial.

High channel contention also leads to variable delays in transmitting a packet (a transmission may succeed without collisions sometimes, but may require several unsuccessful attempts and multiple backoffs some other time). For example, Fig. 9 shows the highly variable TCP RTT of a single client in the controlled experiment with 21 clients described above. Sudden RTT variations confuse the TCP's RTO estimation algorithm, and may lead to TCP timing out unnecessarily, while the data packet is still in transit. In fact, we collected client-side logs in the experiment above and noticed 61 instances of spurious TCP timeouts and retransmissions (summed across all the clients), where the original transmission and the subsequent retransmission were both received by the client after a long delay. While we could not collect client-side logs in the classroom, we did notice highly variable RTTs and expect several of the TCP retransmissions seen were in fact unnecessary.

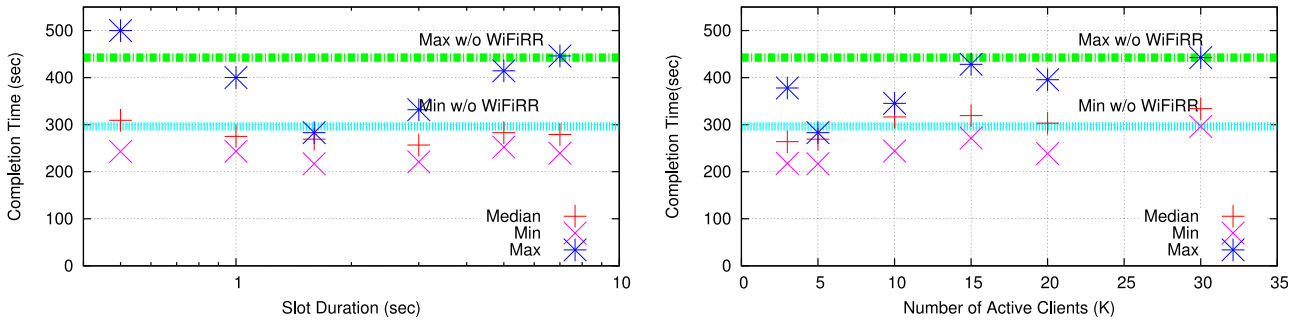
Finally, we could reproduce the phenomenon of client device drivers becoming non-responsive under high contention with several different laptops and device drivers in our controlled experiments as well.

Note that we have used our controlled experiments to verify that there were no other causes of packet loss introduced by the wired channel or the AP in our measurements. We repeated our experiments with APs from two popular vendors, and obtained similar results. We also verified that there was no buffer mismanagement at the AP. For example, with vendor supported AP logs we verified that the AP buffer always had enough packets to transmit over the wireless link, and that buffer underflow was not the reason for poor throughput.

5 Design of WiFiRR

We now describe our solution WiFiRR, that seeks to improve the performance of large TCP downloads in a dense WiFi setting like classrooms.

The measurements and analyses of the previous sections lead us to conclude that high channel contention is root cause for poor TCP download performance. To address this problem, we seek to limit the number of clients contending for the wireless channel at any instant by modifying the AP's MAC-layer scheduling policy. Our solution, WiFiRR, is designed as a modification to the AP. Normally, APs transmit packets belonging to all clients from its buffer in a FIFO manner. With WiFiRR, an AP designates a subset of K out of the total N clients as "active" during a given time slot T. Whenever the AP gets a chance to transmit, the AP looks through its queue and preferentially picks packets to these K active clients, skipping over packets from non-active clients in the queue. The AP varies the set of K active clients



(a) Varying T (K fixed at 5)

(b) Varying K (T fixed at 1.6s)

Fig. 10. Min, median, and max download completion times varying T and K.

in a round-robin fashion in every slot, so that every client eventually gets a chance to make progress. The AP transmits broadcast and management frames normally, as per their turn in the queue. Of course, the AP does not keep the link idle: if there are no broadcast frames or frames to active clients, it will transmit frames to non-active clients.

Now, in a time slot T, since we suppress downlink TCP data and ACK packets for the marked inactive clients, their TCP flows will go quiet for the duration of the slot, and the uplink traffic from the clients is greatly suppressed as well. This leads to a lower number of contending nodes, fewer collision losses, and eventually, better TCP performance for the active clients. Our solution does not require any change at the clients. While our solution leads to a temporary stalling of the non-active clients, the overall performance improves over large TCP downloads, because the clients will experience much better network conditions once they become active.

Intuitively, number of active clients K should be a reasonable value such that the collision on the channel is less. Also, slot-time T should be large enough so that few frames of active clients could be exchanged and should be small enough so that non active clients' performance does not get affected. In our evaluation (Section 6), we show that an empirical value of $K = 5$ and $T = 1.6$ sec works best for a variety of settings.

The principle of WiFiRR is: by suppressing the downlink data and TCP ACKs for uplink data, it expects that the uplink traffic will be reduced and thus the uplink contention will reduce too. But if the uplink traffic can continue without TCP ACKs, then WiFiRR will be less effective. For example, for opening a webpage typical browsers open 6-8 multiple parallel TCP connections for downloading multiple web objects embedded in the webpage. So, from each of the TCP connections TCP SYN packet will be sent by the client. Now if we suppress TCP SYN+ACK packet at the AP, that will result in reattempt of TCP SYN packet transmission by the client. SYN max retry limit is six, so if all the TCP SYN+ACK packets are suppressed, each client will send 6-8 connection requests and six retry requests per connection. This will increase contention in the network. While WiFiRR shows some performance improvement in such scenarios, performance will improve more if a client-side solution is used in conjunction.

As we discussed in Section 2, SPDY [12] is an application layer protocol as a replacement for HTTP. The goal of SPDY is to reduce web page load latency and improve security. Out of different features that SPDY provides, the most important one in our context is TCP stream multiplexing. SPDY opens a single TCP connection for downloading multiple web objects, as opposed to typical HTTP browsers which open 6-8 multiple parallel connections to download multiple web objects. SPDY also allows requests for multiple web objects to be interleaved on a single connection. Thus SPDY reduces the chattiness of the clients. This helps in reducing the contention/collision on the channel. Note that while overall performance improves with client-side support, such support is not necessary for WiFiRR to be beneficial.

A possible concern with WiFiRR as described above is the following. If we suppress traffic for non active flows, this might affect short/interactive flows. Here the negative effects of stalling may overweigh the benefits. To address this in WiFiRR, we do not suppress the non active flows totally, instead we allow a low rate traffic to continue for the non active flows. We show in our evaluation that this is effective in practice.

6 Evaluation

We have done extensive evaluation of WiFiRR in simulation and in real settings. In Section 6.1 we discuss simulation results and in Section 6.2 we discuss experimental results.

6.1 Simulations

We have implemented WiFiRR in ns-3 by modifying AP's MAC layer scheduling policy as described before. We use the same simulation scenario discussed in Sec. 4.1 (Table 2) to evaluate our scheme: a simulation of 30 clients performing a large download (of 5 MB each), while simultaneously generating a low rate upload traffic emulating browser behaviour. Each client opens two connections and sends request for 20 objects with average upload rate at 10 kbps.

We experimented with different values of slot duration T and number of active clients K. Fig. 10a shows the minimum, median, and maximum download completion times over 30 clients with WiFiRR. The slot duration is varied from 500 millisec to 7 sec, and the number of active clients K is fixed at 5. The horizontal lines also show the minimum and maximum completion times without WiFiRR. We find that the worst case completion time reduces to 284 sec from 443 sec (1.6× reduction) when the slot duration is 1.6 sec. Note that the worst case completion time is an important metric in settings like classrooms: the instructor can move

TABLE 3
WiFiRR Evaluation: Browsing Upload

Number of connections	Number of web objects	WiFiRR gain
2	20	1.6×
3	13	1.11×
4	10	1.05×

on to the next activity only after the last student is done. The average RTT of the TCP flows in our simulation was around 400 millisec due to contention, resulting in a slot duration of 1.6 sec working best.

Fig. 10b shows a similar comparison of completion times, where we set the slot size T to 1.6 s, but vary the number of active clients K. Here, $K = 5$ leads to maximum reduction ($1.6\times$) of worst case completion time.

In the above browsing upload model, we vary the number of parallel connections, number of web objects keeping the request and response size and upload rate same i.e., 10 Kbps, and evaluate with WiFiRR. Table 3 shows improvement for different combinations. With more parallel connections, WiFiRR's mechanism of suppressing the downlink ACK is less effective in reducing uplink data traffic.

Next, we discuss results of browsing upload case using SPDY as application layer protocol instead of HTTP. We implemented SPDY in ns-3. We created two application modules and name them as *SPDYClient* and *SPDYServer*. Out of all the features SPDY provides: multiplexing, compression, prioritization and server push; we implemented only the multiplexing i.e., single TCP connection feature as this is the most useful feature for our context. *SPDYClient* opens a single TCP connection and sends requests for 40 objects to the *SPDYServer*. The size of the request and responses are same as of our *ChattyClient* and *WebServer* module (see Section 4.1).

We evaluate SPDY in the same simulation scenario of 30 clients doing a 5 MB download, described before in Table 2. In Fig. 11, we show a comparison of minimum, median and maximum download completion time for the clients for HTTP (base case) and SPDY with and without WiFiRR. SPDY alone improves worst case completion time by a factor of $1.8\times$. Then we evaluate the same scenario after enabling WiFiRR at the AP. SPDY+WiFiRR provides the maximum improvement of worst case completion time by a factor of $2.9\times$.

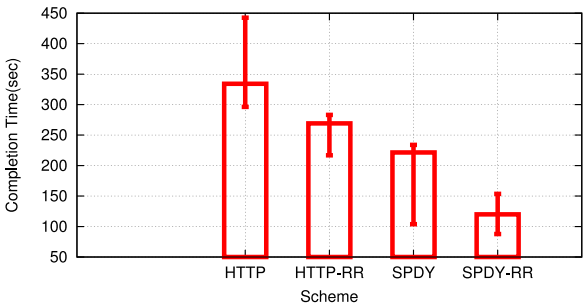


Fig. 11. Min, median, and max download completion times for HTTP and SPDY with & w/o WiFiRR.

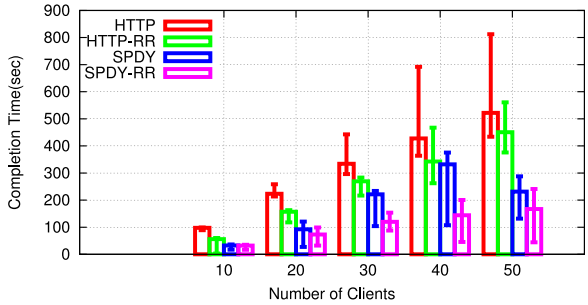


Fig. 12. Collision percentage during download with and w/o WiFiRR.

WiFiRR benefits most from SPDY over HTTP because single TCP connection not only reduces the total uplink traffic but also the contention. When at the AP, WiFiRR marks a client as inactive, packets for that client are queued up till its turn comes. For SPDY all these packets belong to the single connection and TCP adapts to this stalling, suppresses uplink traffic correctly. Now for HTTP as there are multiple TCP connections, each of those connections will adapt to this stalling differently and thus the uplink traffic may not be suppressed adequately.

Now, we inspect whether WiFiRR has indeed reduced the collision percentage. Fig. 12 shows the collision percentage for the base case and with SPDY+WiFiRR. WiFiRR leads to a lower rate of collisions. While the average rate of collisions on the channel was 15-20 percent without WiFiRR, with WiFiRR (1.6 sec slot and 5 active clients) it reduces to 2-3percent.

Next, we examine the scalability of WiFiRR by evaluating it under different test settings. In the above simulation scenario in Table 2, we vary the total number of clients from 10 to 50 and evaluate with WiFiRR. Fig. 13 shows min, median and max download completion time for HTTP and SPDY with and without WiFiRR as the network size grows. WiFiRR provides an improvement of $2.7\times$ for a network size of 10 and $3.4\times$ for a network size of 50. The improvement of WiFiRR does not degrade with increasing number of clients. Thus, WiFiRR scales well with number of clients. WiFiRR provides maximum improvement of $3.5\times$ when the network size is 40.

We also evaluate WiFiRR with another type of upload traffic i.e., constant upload traffic. We have used the same simulation scenario of 30 clients performing a large download (of 5 MB each) described in Table 2. Here we have used the *OnOffApplication* application module in ns-3. Each client opens a single TCP connection and uploads constantly at 10 Kbps rate. We have made the *OnOffApplication*

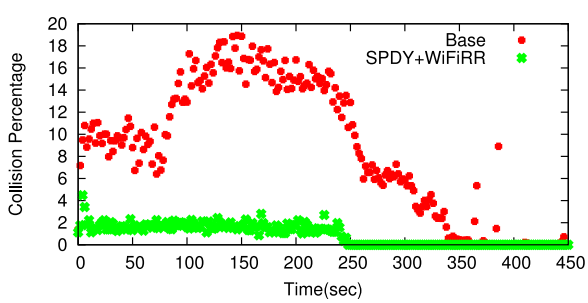


Fig. 13. Scalability of WiFiRR.

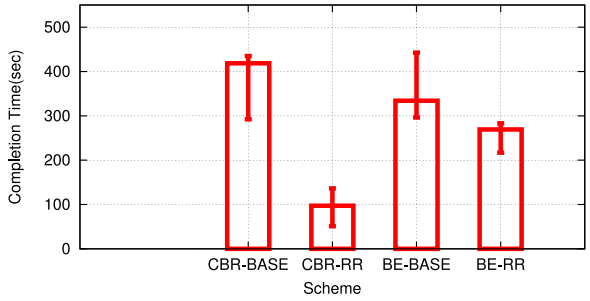


Fig. 14. Min, median, and max download completion times for different upload traffic types.

to be always on and uploads at a rate of 10 Kbps. Fig. 14 shows the min, median and max download completion time for constant upload and browsing upload case with and without WiFiRR. We denote constant upload as CBR (constant bit rate) and denote w/o WiFiRR as CBR-BASE and with WiFiRR as CBR-RR. We denote browsing upload as BE (browsing emulation) and denote w/o WiFiRR as BE-BASE and with WiFiRR as BE-RR. For constant upload case, WiFiRR provides $3.2\times$ reduction in worst case completion time.

Next, we compare our solution WiFiRR (with 1.6 sec slot duration, five active clients) to WiFox [5], another solution that aims to improve the TCP download performance in dense scenarios. WiFox addresses the asymmetry between uplink and downlink traffic by prioritizing the AP (see Section 2). We implemented WiFox in ns3, as per the specification in [5]. We made the AP to use 802.11e, and implemented the high and default priority channel access settings as described in [5]. In WiFox, time is divided into intervals of size T', each of which is divided into n slots. In every time unit T', the AP accesses the channel in a prioritized fashion for $k \leq n$ slots. The mapping between the priority level k and the queue size of the AP can be logarithmic, exponential, linear and logistic. We use the logistic mapping as it gave the best results for WiFox. We also tuned the maximum queue size (from 50 mentioned in the paper to 512) as it resulted in better completion time for WiFox in our simulation setting. We set T' to 100 millisec, n to 10 slots, and set the k range to be 0-10. We evaluate WiFox for both constant upload traffic and browsing upload traffic. Fig. 15 shows min, median and max download completion times with WiFox, as compared to the cases with WiFiRR and without WiFiRR for constant upload traffic. While WiFox does lead to improved performance over the base case without WiFiRR, WiFiRR outperforms WiFox by $2.25\times$ in our scenario of large TCP downloads. For browsing upload traffic, WiFox does not provide any improvement over base case while WiFiRR provides $2.9\times$ improvement.

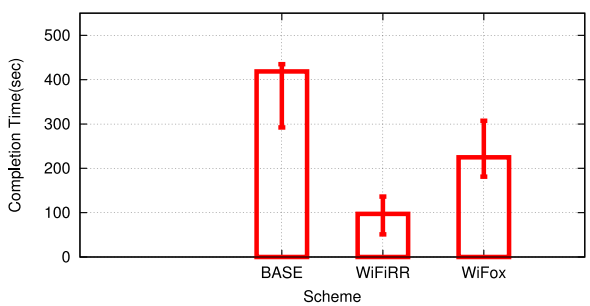


Fig. 15. Min, median, and max download completion times for WiFiRR vs. WiFox.

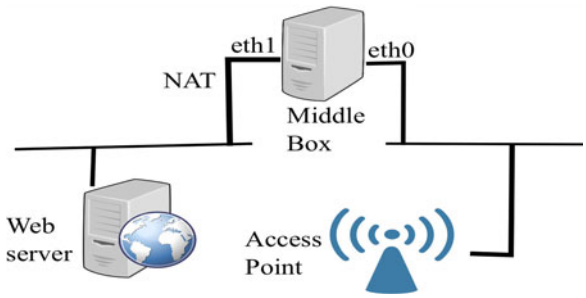


Fig. 16. Mimicking WiFiRR for experimentation.

6.2 Experimental Results

Next, we evaluate WiFiRR in real settings. Prior to experimental result, we first discuss our implementation options.

6.2.1 *Implementation Options*

WiFiRR can be implemented either by modifying AP's driver or at a *middlebox* before the AP. WiFiRR is agnostic to where it is implemented. Given that every vendor might not allow modifications to the driver to support WiFiRR, implementing it at a middlebox before the AP is a more practical solution. So, instead of modifying AP's MAC layer scheduling policy, we implemented WiFiRR as a scheduler before the AP. The setup is shown in Fig. 16.

The middlebox has two ethernet interfaces connected to two different subnets. The first one (eth0) is connected to the AP's subnet and the other one (eth1) is to the web server's subnet. In the middle box, we forward traffic from one interface to the other. We perform source natting at eth1 so that all the traffic forwarded by eth0 can reach the webserver's subnet. Now all the traffic to and from the access point will go via the middlebox.

We implemented WiFiRR using Linux *tc*. We created classful queuing discipline (qdisc) at eth0 i.e., the ethernet interface connected to the AP's subnet. We used HTB i.e., hierarchical token bucket queuing discipline.

Fig. 17 shows the schematic diagram of WiFiRR implementation using HTB. Under root qdisc of the ethernet interface we created child classes, each of which has a rate and ceiling parameter. Rate specifies the minimum bandwidth a class is assigned, ceiling specifies the maximum

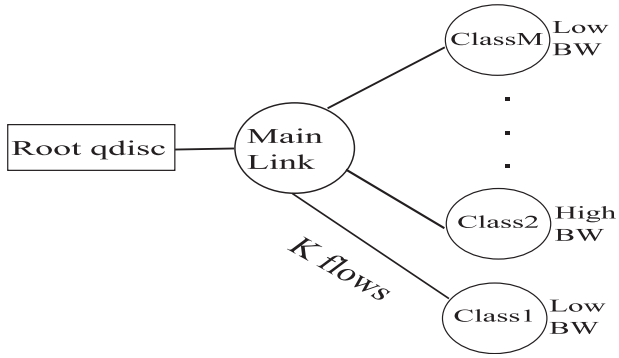


Fig. 17. WiFiRR implementation using HTB.

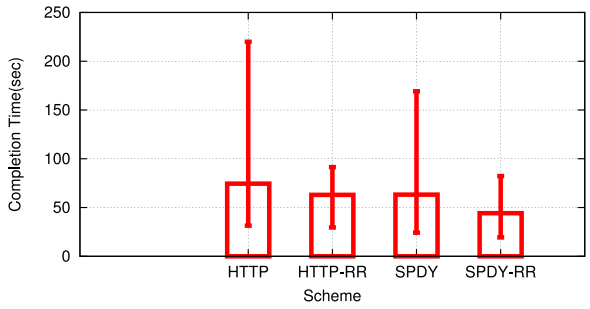


Fig. 18. Min, median, and max download completion times for HTTP and SPDY with & w/o WiFiRR.

bandwidth a class can use. The algorithm is as follows: we mapped a set of K flows to one class, so for total N flows there are $M = N/K$ classes. Then in round-robin fashion, we provided high bandwidth configuration to one class and low to others. The round-robin frequency is T time. In implementation, we have used $K = 5$ flows and $T = 1.6$ sec.

6.2.2 Results

Now we discuss experimental results. We emulate the browsing behaviour as observed in the classroom by using Epload tool [11]. The Epload tool emulates the page load process by segregating network operations from computations. This is a replacement for browser. In Epload, computations are performed deterministically, which gives a measure of repeatability to the experiments. Epload works as follows: it first records the "dependency graph" of an webpage using WProf [20]. The dependency graph captures the computations and network operations to be made, timings of those and their dependencies. The Epload replays the dependency graph i.e., the computations and network operations for the page load process. Epload supports HTTP, HTTPs (opens 6-8 multiple parallel connections) and SPDY (opens single connection) as the application layer protocol.

To evaluate WiFiRR, we set up an experiment with 15 laptops connected to one 802.11n access point. Prior to the experiment, we ensured that the wired backhaul, server and clients are not the bottleneck. We also switched off other BSS working on same/neighboring channel of ours during the experiment. We performed experiment at night to avoid any interference on 2.4 Ghz band. We collected traces at our custom instrumented AP collecting per-frame MAC layer statistics and tcpdump at the server. Along with the browsing upload, all the clients download a 9 MB file.

Fig. 18 shows min, median and max download completion time for the clients for HTTP and SPDY with and without WiFiRR. For base case (i.e., HTTP), the worst case completion time is 220 sec. Then we enabled WiFiRR at the middlebox, it reduces the worst case completion time to 91.44 sec, the improvement over base case is 2.4×. After that, we experimented with SPDY as the application layer protocol instead of HTTP. We installed SPDY module i.e., *mod_spdy* module for apache at the server, this enables SPDY at the server. At the client side, we used Epload module with SPDY as the application layer protocol instead of HTTP. We configured SPDY by disabling SSL: that eases debugging. SPDY alone reduces the worst case completion time to 169 sec, the improvement over HTTP is 1.3×. Finally, we enabled WiFiRR at middlebox and performed the same experiment with SPDY protocol. The worst case completion time with SPDY+ WiFiRR is 82 sec, the improvement over base case (HTTP) is 2.7×. SPDY +WiFiRR combination provides the most improvement, as expected.

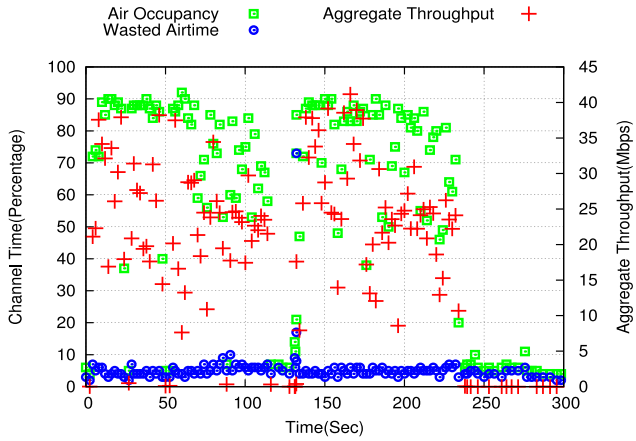


Fig. 20. Aggregate throughput, air occupancy, and wasted airtime at the AP with WiFiRR.

Now we look at different metrics discussed in Section 3 to understand how WiFiRR helps in improving application layer performance. We first look at the TCP retransmission percentage. Fig. 19 shows average retransmission rate (averaged across all clients every 2 s) as a function of time. The average TCP retransmission percentage with WiFiRR is 2-5 percent. This explains better performance of large TCP downloads with WiFiRR.

Next, we inspect impact of WiFiRR on MAC layer performance. We specifically see air occupancy, wasted airtime and aggregate throughput. The corresponding metrics are shown in Fig. 20. Here too, air is busy for most of the times but now aggregate throughput has improved: it is between

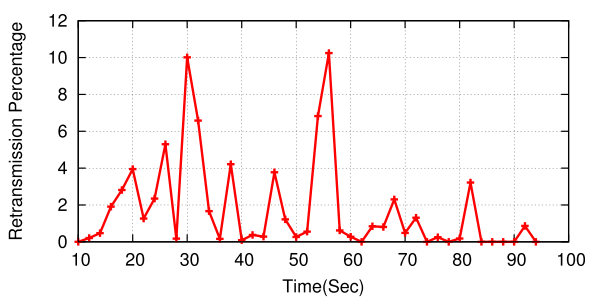


Fig. 19. The average TCP retransmission rate across all clients vs. time. with WiFiRR.

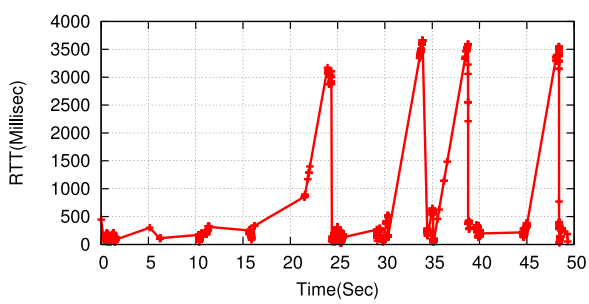


Fig. 21. RTT with WiFiRR.

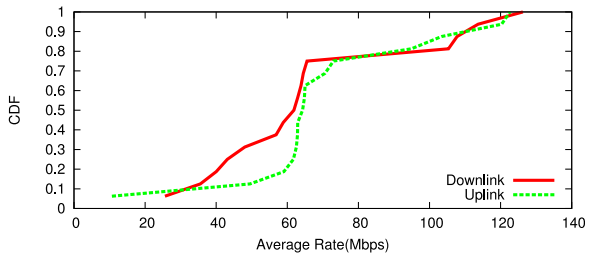


Fig. 22. CDF of the time-averaged bit rates of clients in the uplink and down link directions with WiFiRR.

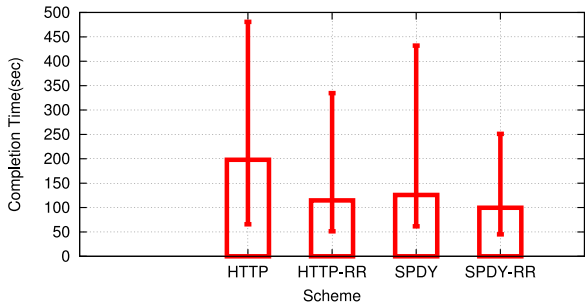


Fig. 23. Min, median and max download completion times for HTTP and SPDY with & w/o WiFiRR in presence of neighboring APs.

$20-40$ Mbps. The airtime wasted due to contention is also small (less than 8 percent).

Now, we look at TCP RTT with WiFiRR. Fig. 21 shows RTT of a sample client after enabling WiFiRR. There is a saw-tooth pattern for RTT. This is expected, because WiFiRR suppresses inactive flows in view of better performance for active flows. So during slot time T, inactive flows' packets will not be sent by the AP. Thus a flow's packet might have to sit at the AP's queue for few times the slot duration till its turn comes. Eventually when the flow becomes active RTT reduces. TCP's RTO estimation algorithm is able to adapt to these changes and does not create any performance problem.

Next, we look at average bit rate of the clients. Fig. 22 shows CDF of time averaged physical layer bit rate of the clients in uplink and downlink directions after enabling WiFiRR. The clients are operating at high bit rate.

To understand the impact of WiFiRR in presence of multiple APs, we use the same experimental setup described before. 15 laptops were connected to one 802.11 n access point. It was set to operate at channel 6 of 2.4 GHz band. But unlike the previous scenario (i.e., no neighboring APs), there were two other access points operating on channel 6. We placed our operating AP close to these access points. From our custom instrumentation of the AP, we observed around 30 percent of the airtime was taken up by these other access points throughout the experiment. Fig. 23 shows min, median and max download completion time of the clients for HTTP and SPDY with and without WiFiRR. For base case (i.e., HTTP), the worst case completion time is 481 sec. Note that, here the completion time is higher than the previous single AP case as the interference from neighboring APs reduces the operating AP's airtime. WiFiRR provides an improvement of $1.3\times$ over HTTP. SPDY alone provides an improvement of $1.1\times$ over HTTP. Finally, SPDY + WiFiRR provides an improvement of $1.9\times$. Here the improvement is lesser than the previous single AP case ($2.7\times$), because of the interference from neighboring APs.

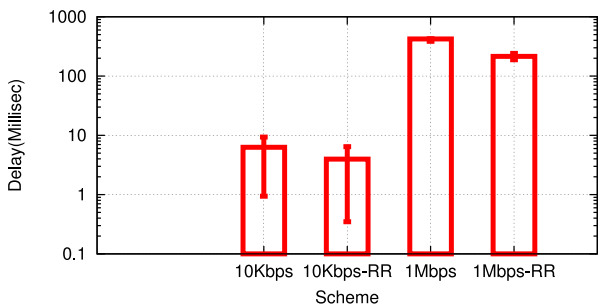
6.3 Interactive Traffic

To understand the effect of WiFiRR on interactive application, we evaluated WiFiRR for different interactive applications: using simulations as well as real experiments.

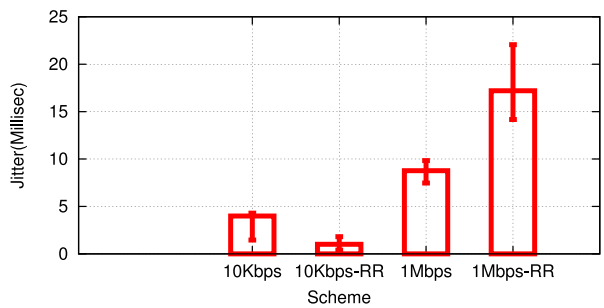
6.3.1 *Simulations*

We used the same simulation scenario of 30 clients connected to one 802.11g access point. To simulate interactive traffic, 30 clients perform constant download from the server. Here we have used the existing *OnOffApplication* application module in ns-3. We have evaluated WiFiRR for low (10 Kbps for e.g., Skype audio) and high (1024 Kbps for e.g., video traffic) download rates.

Here we measure delay and jitter, delay is one way time and jitter is variation of the delay at client. Figs. 24a and 24b show the min, median and max of delay and jitter across the flows for 10 and 1024 Kbps upload rate with and without WiFiRR. For 10 Kbps upload rate, the delay profile with and without WiFiRR is similar, between 1-9 ms. The jitter for base case i.e., without WiFiRR is between 1-4.5 ms, and with WiFiRR it reduces to 0.4 ms-2 ms. One might wonder how the delay is in order of few ms, where as the slot time T for WiFiRR is itself few hundreds of ms. This is due to the fact that we allow a low rate traffic to continue for the non active flows. For 1024 Kbps upload rate, the delay without WiFiRR is between 380-430 ms, with WiFiRR it reduces to 200-240 ms. The jitter without WiFiRR is between 7-10 ms, with WiFiRR it increases to 14-22 ms. Thus, WiFiRR improves delay, while causing only a minor increase in the jitter; note that jitter values of up to about 30 ms are considered acceptable for interactive traffic [21].



(a) Delay



(b) Jitter

Fig. 24. Interactive traffic: Min, median, and max delay, jitter with and w/o WiFiRR.

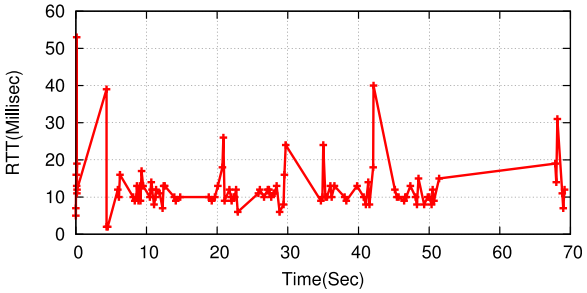


Fig. 25. SSH: RTT with WiFiRR.

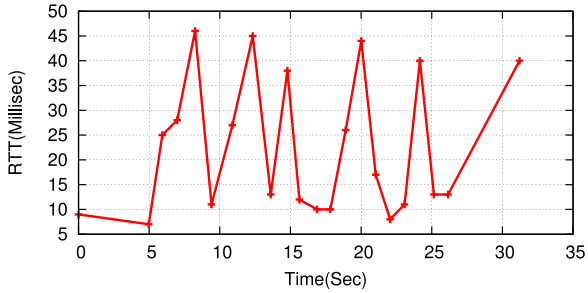


Fig. 26. Chat: RTT with WiFiRR.

6.3.2 Experimental Result

In real settings, we evaluated WiFiRR for SSH (secure shell) and chat applications. The same experimental set up was used like in Section 6.2 i.e., 15 clients were connected to one 802.11n AP. WiFiRR was enabled in the middlebox. We put a lower minimum bandwidth for each classes in the HTB. The ceiling parameter of the HTB classes was specified as 100 Kbps, so the non active flows can take up to 100 Kbps bandwidth of the link. We collected traces at our custom instrumented AP collecting per-frame MAC layer statistics, and tcpdump at the server and at the clients.

SSH traffic. Here all the clients remotely logged into a server using ssh, and ran different commands. At the user side we did not notice any perceptible delay. We measure RTT at the server. RTT for this traffic is shown in Fig. 25. It is less than 40 ms which is imperceptible to ssh users. As we allow a low rate traffic to continue, WiFiRR could successfully support low bit rate interactive traffic. The minimum bandwidth to other non active flows did not cause any performance impact on the active flows.

Chat application. We used the same setup for testing a chat application. All the clients run a chat application, while the chat server runs on the server. We did not notice any perceptible delay while entering the message. The RTT for the chat application is shown in Fig. 26. It is less than 45 ms, which is too small for chat users to notice.

7 Conclusion

This paper presented measurements of TCP download performance in a dense WiFi scenario of WiFi-enabled classroom, where students download quizzes and instruction material over WiFi. Our results show that TCP download performance degrades significantly with increased user density, much more beyond what is to be expected from prior work. We analyze the reason for this poor performance and find that the small amount of background upload traffic that coexists with the TCP download traffic in real life causes an increase in contention on the wireless channel. The subsequent collision losses trigger undesirable behavior in other protocols: the bit rate adaptation unnecessarily lowers its bit rate, TCP gets confused by the highly variable RTTs and performs spurious retransmits, and device drivers perform unexpectedly under such losses. We then propose a solution, WiFiRR, that improves the performance of large TCP downloads in a dense scenario. Our solution operates as a scheduler at the AP, and restricts the number of active clients contending for the channel at any instant by selectively transmitting packets to different subsets of active clients over different slots. To reduce the client side chattiness even more, we then incorporate SPDY into our solution. SPDY opens single connection for multiple web objects thus reduces chattiness. We have done extensive evaluation of WiFiRR in simulation and in real experiments. WiFiRR provides most improvement when tried with SPDY compared to HTTP. For a sample scenario of 30 clients, in simulation, HTTP+WiFiRR reduces worst case download completion time by $1.6\times$ and SPDY+WiFiRR by $2.9\times$. In real settings, HTTP+WiFiRR reduces worst case download completion time by $2.4\times$ and SPDY+WiFiRR by $2.7\times$. WiFiRR scales well irrespective of the network size. The maximum gain with WiFiRR is $3.5\times$. We have also evaluated WiFiRR for interactive traffic. WiFiRR does not harm interactive traffic's performance since we allow a low rate traffic to continue from the non active flows.

This paper thus examines and effectively addresses performance problems in emerging use cases of WiFi: dense settings with several tens or more clients, such as conferences, sports stadiums, and WiFi-enabled classrooms. Going forward, we plan to analyze mathematical foundation of WiFiRR by following the Bianchi's model.

References

[1] R. Bruno, M. Conti, and E. Gregori, "Modeling TCP throughput over wireless LANs," in *Proc. 17th IMACS World Congr. Sci. Comput., Appl. Math. Simul.*, 2005, pp. 11–15.

[2] G. Kuriakos, S. Harsha, A. Kumar, and V. Sharma, "Analytical models for capacity estimation of IEEE 802.11 WLANs using DCF for internet applications," *Wireless Netw.*, vol. 15, no. 2, pp. 259–277, 2009.

[3] M. A. Ergin, K. Ramachandran, and M. Gruteser, "An experimental study of inter-cell interference effects on system performance in unplanned wireless LAN deployments," *Comput. Netw.*, vol. 52. no. 14, pp. 2728–2744, 2008.

[4] S. Choi, K. Park, and C.-k. Kim, "On the performance characteristics of WLANs: Revisited," in *Proc. SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 2005, pp. 97–108.

[5] A. Gupta, J. Min, and I. Rhee, "WiFox: Scaling WiFi performance for large audience environments," in *Proc. 8th Int. Conf. Emerging Netw. Experiments Technol.*, 2012, pp. 217–228.

[6] E. Lopez-Aguiler, J. Casademont, J. Cotrina, and A. Rojas, "Performance enhancement of WLAN IEEE 802.11 fot asymmetric traffic," in *Proc. Int. Symp. Pers., Indoor Mobile Radio Commun.*, 2005, pp. 1463–1467.

[7] X. Wang and S. A. Mujtaba, "Performance enhancement of 802.11 wireless LAN for asymmetric traffic using an adaptive MAC layer protocol," in *Proc. IEEE 56th Veh. Technol. Conf.*, 2002, pp. 753–757.

[8] D. Malone, D. J. Leith, A. Aggarwal, and I. Dangerfield, "Spurious TCP timeouts in 802.11 networks," in *Proc. 6th Int. Symp. Model. Optimization Mobile, Ad Hoc, Wireless Netw. Workshops*, 2008, pp. 43–49.

[9] P. A. K. Acharya, A. Sharma, E. M. Belding, K. C. Almeroth, and K. Papagiannaki, "Congestion-aware rate adaptation in wireless networks: A measurement-driven approach," in *Proc. 5th Annu. IEEE Commun. Soc. Conf. Sensor, Mesh Ad Hoc Commun. Netw.*, 2008, pp. 1–9.

[10] K. V. Cardoso and J. F. de Rezende, "Increasing throughput in dense 802.11 networks by automatic rate adaptation improvement," *Wireless Netw.*, vol. 18, no. 1, pp. 95–112, 2012.
[11] (2014). [Online]. Available: http://wprof.cs.washington.edu/spdy/tool/
[12] (2009). [Online]. Available: https://www.chromium.org/spdy/spdy-whitepaper
[13] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *IEEE J. Select. Areas Commun.*, vol. 18, no. 3, pp. 535–547, Mar. 2000.
[14] A. Kumar, E. Altman, D. Miorandi, and M. Goyal, "New insights from a fixed-point analysis of single cell IEEE 802.11 WLANs," *IEEE/ACM Trans. Netw.*, vol. 15, no. 3, pp. 588–601, Jun. 2007.
[15] (2016). Status of project IEEE 802.11ah [Online]. Available: http://www.ieee802.org/11/Reports/tgah_update.htm
[16] Y. Yuan, W. A. Arbaugh, and S. Lu, "Towards scalable MAC design for high-speed wireless LANs," *EURASIP J. Wireless Commun. Netw.*, vol. 2007, 2007, Art. no. 12597.
[17] Z. Z. Abichar and J. M. Chang, "Group-based medium access control for IEEE 802.11 n wireless LANs," *IEEE Trans. Mobile Comput.*, vol. 12, no. 2, pp. 304–317, Feb. 2013.
[18] Y. Yang and S. Roy, "Grouping-based MAC protocols for EV charging data transmission in smart metering network," *IEEE J. Select. Areas Commun.*, vol. 32, no. 7, pp. 1328–1343, Jul. 2014.
[19] (2011). Potential compromise for 802.11ah use case document, IEEE 802.11-11/0457r0, 2011, IEEE 802.11ah TG.
[20] (2014). [Online]. Available: http://wprof.cs.washington.edu/
[21] (2005). Enterprise QoS solution reference network design guide [Online]. Available: http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/WAN_and_MAN/QoS_SRND/QoS-SRND-Book.pdf

Mukulika Maity received the BE degree in computer science and engineering from Bengal Engineering and Science University, Shibpur, India, in 2010. Then, she joined the Computer Science and Engineering Department at the Indian Institute of Technology, Bombay, India, in 2010 to pursue her MTech degree and in 2012, she converted to the PhD degree. Her PhD topic is health diagnosis and congestion mitigation of WiFi networks. Her research interests are broadly in the area of wireless networks and mobile computing.

Bhaskaran Raman received the BTech degree in computer science and engineering from the Indian Institute of Technology, Madras, in May 1997. He received the MS and PhD degrees in computer science from the University of California, Berkeley, in 1999 and 2002, respectively. He was a faculty in the CSE Department at the Indian Institute of Technology Kanpur, Kanpur, India, from June 2003. Since July 2007, he has been a faculty at the CSE Department at the Indian Institute of Technology, Bombay, India. His research interests include communication networks, wireless/mobile networks, large-scale Internet-based systems, and Internet middleware services.

Mythili Vutukuru received the BTech degree in computer science and engineering from the Indian Institute of Technology, Madras, in 2004. She received the MS and PhD degrees in computer science from the Massachusetts Institute of Technology in 2006 and 2010, respectively. After her PhD, she worked at Movik Networks, a startup in the telecom space, for 3 years. Since July 2013, she has been a faculty at the CSE Department at the Indian Institute of Technology, Bombay (India). Her research interests are in networked systems, wireless communication, and network security.