Received March 24, 2018, accepted May 3, 2018, date of publication May 15, 2018, date of current version June 5, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2836439

Battling Latency in Modern Wireless Networks

AHMAD SHOWAIL[1,2], AND BASEM SHIHADA[1], (Senior Member, IEEE)
[1]Computer, Electrical and Mathematical Sciences and Engineering Division, King Abdullah University of Science and Technology, Thuwal 23955-6900, Saudi Arabia
[2]Computer Science and Information Technology College, University of Prince Mugrin, Medina 41499, Saudi Arabia

Corresponding author: Basem Shihada (basem.shihada@kaust.edu.sa)

ABSTRACT Buffer sizing has a tremendous effect on the performance of Wi-Fi based networks. Choosing the right buffer size is challenging due to the dynamic nature of the wireless environment. Over buffering or 'bufferbloat' may produce unacceptable end-to-end delays. On the other hand, small buffers may limit the performance gains that can be obtained with various IEEE 802.11n/ac enhancements, such as frame aggregation. We propose wireless queue management (WQM), a novel, practical, and lightweight queue management scheme for wireless networks. WQM adapts the buffer size based on the wireless link characteristics and the network load. Furthermore, it accounts for aggregates length when deciding on the optimal buffer size. We evaluate WQM using our 10 nodes wireless testbed. WQM reduces the end-to-end delay by an order of magnitude compared to the default buffer size in Linux while achieving similar network throughput. Also, WQM outperforms state of the art bufferbloat solutions, namely CoDel and PIE. WQM achieves seven times less latency compared to PIE, and two times compared to CoDel at the cost of 8% drop in goodput in the worst case. Further, WQM improves network fairness as it limits the ability of a single flow to saturate the buffers.

INDEX TERMS Bufferbloat, IEEE 802.11, frame aggregation, A-MPDU, TCP.

I. INTRODUCTION

Overbuffering is becoming common in today's data networks. While big buffers may potentially help in limiting packet drops, they do not come for free. In fact, large buffer sizes may result in high end-to-end latency. Overbuffering or 'bufferbloat' [1], [2] is responsible for long delays in the Internet. These delays could be in the order of seconds. As computer memory is becoming cheaper with time, more people are suffering from performance degradation caused by large buffers. The fallacy that 'more is always better' is what made end-user equipment less efficient.

It is challenging to tackle bufferbloat in wireless networks. One of the main reasons is the variable link capacity in such a network. With the help of rate control mechanisms, the link speed may be altered based on various parameters such as the level of interference and the distance between the sender and the receiver. Just to give an example, the variation in link speed in WiFi networks could be large as two orders of magnitude. Hence, static buffer sizes may result in a sever degradation in network performance. Another reason is the shared nature of the wireless medium which is true for all kinds of wireless networks. As a result, nodes in the same network are going to contend for wireless channel access. This will cause the actual link capacity for each node to be less than the physical capacity. Therefore, the amount of buffer in each client must to be set according to the actual link rate. Finally, wireless networks suffer from high variance in packet interservice rate because of the large number of corrupted and lost packets that will get eventually retransmitted.

MAC-layer frame aggregation is one of the enhancements made in IEEE 802.11n/ac standard specifications to improve the performance of the wireless network. This feature enables the wireless node to send multiple frames at the same time. In fact, the exact scheduler logic is not specified in the standard specifications and hence each vendor might implement it in a different way. Transmitting full length aggregates may maximize the throughput, however, it increases the delay since the sender needs to wait for the assembly of all subframes from higher layers. One way to reduce this delay is by transmitting whatever frames available in the queue in a timely manner. As a result, the length of the aggregates is going to vary over time. This variability in aggregate length poses a new challenge to accurately estimate the queue draining time based on the current transmission rate. Other enhancements in IEEE 802.11n, such as channel bonding and Multiple-Input and Multiple-Output (MIMO) streams, allow Wi-Fi radios to operate at link rates as high as 600 Mb/s. Thus, there is a huge variation in the queue draining time between the highest and lowest possible rates. For example, assume a single sender and receiver, both configured with the default Linux buffer size of 1000 packets. The 600 Mb/s link needs only 20 ms to drain the buffer; however, this buffer drain time is two orders of magnitude higher when using the 6.5 Mb/s link.

In this paper, we propose Wireless Queue Management (WQM) which is a solution to bufferbloat in the wireless domain. WQM is an aggregation aware buffer management tool for dynamic buffer allocation in WiFi based networks. WQM smartly distinguishes between 'useful' and 'disruptive' buffers. Useful buffers are the ones used to absorb bursty traffic. On the other hand, disruptive buffers are going to increase the end-to-end latency without enhancing the throughput. WQM is considered practical for several reasons. First, it uses a passive measurement technique. Hence, WQM does not impose additional overhead for measurements collection. Second, WQM uses the actual link transmission rate to calculate the buffer drain time. Based on that, it adjusts the buffer size in order to reduce queueing delays while allowing enough buffers to saturate the network. Finally, WQM accounts for frame aggregation when estimating the queue draining time. We implemented and evaluated WQM in a Linux testbed and found that it manages to reduce the latency by an order of magnitude compared to the de facto buffer sizing scheme in recent Linux kernels. Furthermore, WQM reduces the queuing delay by up to $7\times$ when compared to other bufferbloat solutions at the cost of less than 8% drop in goodput.

We believe that this is the first work that attempts to tackle bufferbloat in 802.11n/ac networks. When WQM is compared to other mechanisms in the literature, it is considered more practical as it does not involve time-stamping the packets at their arrival to the queue. Another unique feature of WQM is the fact that it accounts for frame aggregation when calculating the best queue size.

II. PRELIMINARIES

In this section, we provide some related background material that we see essential for introducing WQM.

A. FRAME AGGREGATION

Several new enhancements are introduced in the IEEE 802.11n/ac standard specifications to improve wireless network utilization, including frame aggregation which simply means sending multiple frames back-to-back. Each of these frames is going to have it's own MAC header and Frame Check Sequence (FCS) trailer. This big frame is called Aggregate MAC Protocol Data Unit (A-MPDU). A-MPDU size is limited to 64KB which is bound by the HT-SIG headers. Each A-MPDU is able to transfer upto 64 subframes (limited by the Block Ack (BA) frame). In fact, IEEE 802.11ac devices can send A-MPDUs as big as 1MB. Hence, using static small buffers is infeasible as it may limit the overall network capacity. To solve this issue, WQM forces the buffer size to be at least equal to the maximum number of subframes per aggregate. One major difference between the aggregation scheme in IEEE 802.11n and IEEE 802.11ac is the fact that the latter always sends frames as aggregates even if the sender has only a single frame to send.

B. ACTIVE QUEUE MANAGEMENT

The main goal of Active Queue Management (AQM) techniques is to make sure that there are no large queues at intermediary network hosts. They are able to achieve this goal using proactive and probabilistic packet dropping. In fact, these algorithms are not widely used in practice because it is very difficult to set the configuration parameter knobs for them effectively. In 2012, a no-knobs AQM technique called CoDel [3] was proposed. Unlike traditional AQM techniques, CoDel does not monitor buffer size or queue occupancy directly. Instead, it keeps track of the minimum queue length for a period that is longer than the nominal Round Trip Time (RTT). This is important because the algorithm does not allow packet dropping if the buffer has less than one Maximum Transmission Unit (MTU) bytes. Additionally, CoDel keeps track of the packet sojourn time instead of measuring the buffer size. Hence, it has clear reflection of user experience. Once the latency exceeds the threshold for some predefined period of time, the algorithm enters the dropping phase. It will exit this phase only if the latency goes below the threshold.

Similarly, researchers from Cisco proposed another no-knobs AQM variant, called PIE (Proportional Integral controller Enhanced) [4]. PIE determines the level of network congestion based on latency moving trends. Upon packet arrival, the packet may be dropped according to a dropping probability. This dropping probability is calculated on a periodic basis based on the dequeue rate and the length of the queue. Both PIE and CoDel targets queuing delay directly without necessarily restricting the buffer size. However, unlike CoDel, PIE does not keep track of the per packet timestamp. Moreover, it decides whether or not to drop a packet before actually queuing it.

While neither CoDel nor PIE are specifically designed for wireless networks, simulation results shows that they manage to respond to changes in link rates while achieving a utilization similar to the traditional tail drop approach [3], [4]. This, however, may not be enough to support fast mobility in wireless devices (*e.g.*, vehicular speed mobility). Furthermore, it is unclear how AQM based techniques can be effectively used in multi-hop wireless networks where the bottleneck spans multiple distributed nodes [5]. Finally, both of these schemes never consider frame aggregation in the buffer sizing decision.

C. RATE CONTROL ALGORITHM

Wired link rates are constant and often known apriori. In contrast, link rate adaptation algorithms dynamically set the wireless link rate in response to changing network conditions. Depending on the link rate adaptation algorithm, these link rates may vary on time scales ranging from milliseconds to minutes. This has implications on the network Bandwidth Delay Product (BDP) and the resulting queue size required for saturating the link. The de facto rate control mechanism in recent Linux kernels is called Minstrel [6]. Minstrel uses active measurements in order to choose the optimal link rate. Simply, it transmits packets periodically using static link rates and then chooses the best rate based on the packet transmission success rate.

III. MOTIVATION

In this section, we motivate the idea of adaptive buffer sizing using testbed experimental analysis. We first analyze the performance of the default buffering scheme in Linux. After that, we discuss why static buffering is not a feasible option for WiFi networks. Finally, we show the effect of frame aggregation on the buffer sizing problem.

A. HOW BAD IS THE LATENCY IN TODAY'S NETWORKS?

To highlight the impact of large buffers on network performance, we transfer a large file between two wireless hosts in our testbed while fixing the wireless link capacity. Using static link rates help in understanding the effect of various link rates on network dynamics. Our testbed uses Atheros IEEE 802.11n wireless cards on Linux machines with ath9k [7] drivers. By default, the size of Linux transmit queue, *txqueue*, on recent Linux kernels is 1000 packets. We use a radio channel that does not interfere with our campus production network. The detailed experimental setup is listed in Section V. Fig. 1 shows the sender TCP congestion window, RTT, and egress buffer occupancy using the 6.5 Mb/s link rate. As window scaling is enabled by default in our testbed, TCP congestion window reaches the limit of 1.6 million bytes before experiencing a packet loss. We observe that this single file transfer is able to saturate such a large buffer which results in long queueing delays, with RTT values exceeding 2 seconds. It is worth noting that queue occupancy never drops to zero although the TCP congestion control algorithm (CUBIC by default) halves the congestion window in reaction to buffer over flow multiple times during the experiment. This is a clear example of bad buffers that does not increase the network utilization and only contributes to network latency.

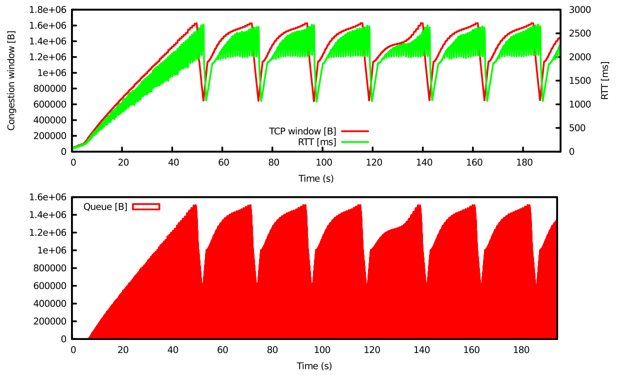

FIGURE 1. TCP congestion window, RTT, and egress queue utilization with a one-hop TCP flow in our testbed with 6.5 Mb/s link rate. Buffer size correspond to values in the stock Linux kernel.

B. HOW DOES FRAME AGGREGATION AFFECT THE BUFFER SIZING DECISION?

A-MPDU aggregation logic is not specified in the standard and is implementation dependant. In fact, there is a trade-off between using the channel resources efficiently and lowering the latency. By using large aggregation frames whenever the quality of the channel is good, we can maximize the channel utilization. On the other hand, one way to minimize the queueing delays is process packets whenever they are ready. A very basic packet scheduler will wait for a full length A-MPDU frame to be assembled before transmitting it which can potentially lead to a very high throughput, but will deteriorate the delay performance. When digging deep in the device driver code, we find that the ath9k [7] scheduler aggregates as many MPDUs as available at that time in the buffer subject to the regulatory and receiver constraints instead of waiting to assemble maximal allowable A-MPDU aggregates which may maximize throughput. To understand the relation between buffer sizing and the level of frame aggregation, we repeat the previous TCP file transfer experiment at multiple link rates and *txqueue* buffer sizes. We report the mean aggregate length for various buffer sizes and link rates in Fig. 2. It is clear from this figure that bigger buffers allow the formation of longer aggregates especially with fast links. This is another evidence that fixing the buffer size to a small value is not always the best thing to do in order to reduce latency. This figure also shows that the link rate directly determines the level of A-MPDU aggregation. Error bars in the figure, which represent maximum and minimum aggregate length, show that the aggregate length varies even with fixed link rates. The only exception is with the 6.5 Mb/s link as transmitting more than one subframe per A-MPDU using the 6.5 Mb/s link is not feasible due to the violation

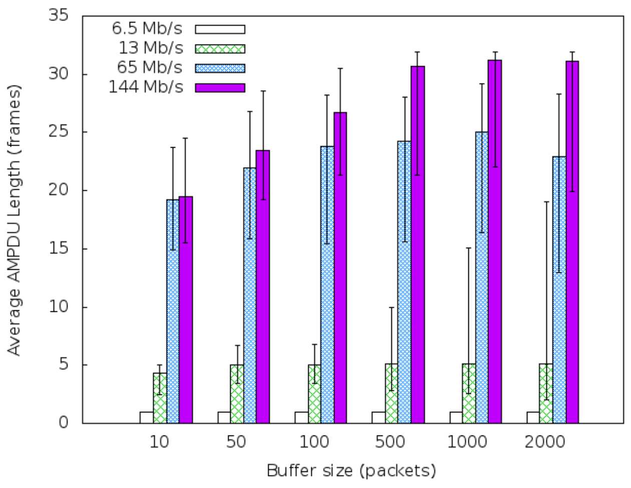

FIGURE 2. Average A-MPDU length of a TCP large file transfer for various link rates and buffer sizes.

of the 4 ms regulatory frame transmission requirement in the 5 GHz band used in our experiments.

IV. APPROACH

In this section, we describe WQM operation and show how we select various WQM parameters.

A. WQM OPERATION

WQM algorithm is described in Algorithm. 1. Clearly, WQM operation pass through two stages. The first stage is called the initial stage and the second stage is called the adjustment stage. WQM is going to calculate the initial buffer size In the initial stage based on a variation of the buffer sizing rule of tuhmb (BDP [8]). In BDP, the buffer should be at least equal to the product of the bandwidth of the bottleneck link with the round trip delay. However, this rule was initially designed for wired networks, and cannot be used directly in Wi-Fi based networks as it does not account for A-MPDU frame aggregation. For example, it is obvious that the transmission time, and hence RTT, for a single frame is less than that of a single A-MPDU. In fact, obtaining the end-to-end delay is not always straight forward. This is why WQM uses only the single-hop RTT to initialize the buffer. Then, it starts adapting the buffer size according to the actual RTT. This mechanism minimizes the queuing delay and maximizes utilization without sacrificing latency for long-running traffic. Hence, the initial buffer could be calculated as:

$$B_{initial} = R * ARTT \tag{1}$$

where $B_{initial}$ is the buffer size that is initially assigned, R is the Tx rate, and $ARTT$ is the round-trip time for a single A-MPDU aggregate as illustrated in Fig. 3. $ARTT$ can be calculated using the equations in Sect. IV-B.

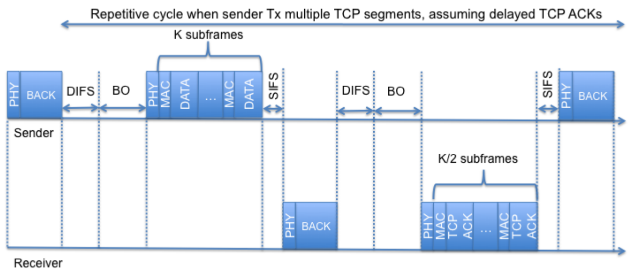

FIGURE 3. **IEEE 802.11n MAC overhead per A-MPDU transmission.**

After assigning the initial buffer size, WQM will start the adjustment phase, where the size of the buffer is adjusted to be synchronized with the load of the network. Periodically, WQM calculates the queuing delay using the following formula:

$$T_{drain} = \frac{(BL/R)}{F} \tag{2}$$

where T_{drain} is the duration of time needed to empty the buffer, BL is the amount of backlog in the queue, and F is

Algorithm 1 WQM Algorithm

```
1:  Fix $limit$ to the acceptable latency
2:  Find the initial buffer size $B_{initial}$ using the Transmission
    rate ($R$) and an A-MPDU RTT ($ARTT$):
3:  $B_{initial} = ARTT * R$
4:  **for** $measurement\ interval$ **do**
5:      Find the buffer draining time $T_{drain}$ using the number
        of bits in the buffer ($BL$) and the duration the channel
        is found to be free ($F$)
6:      $T_{drain} = \frac{BL/R}{F}$
7:      Tune the buffer size $B$ if the network is bloated
8:      **if** $T_{drain} > limit$ and $B > B_{min}$ **then**
9:          **if** $alarm_{high}$ is ON **then**
10:             $B = \max(\lceil B/2 \rceil, B_{min})$
11:         **else**
12:             set $alarm_{high}$ to ON and $alarm_{low}$ to OFF
13:         **end if**
14:     **else if** $T_{drain} < limit$ and $B < B_{max}$ **then**
15:         **if** $alarm_{low}$ is ON **then**
16:             $B = B + 1$
17:         **else**
18:             set $alarm_{low}$ to ON and $alarm_{high}$ to OFF
19:         **end if**
20:     **end if**
21: **end for**
```

the period where the channel is found to be free, *i.e.,* channel is not busy. WQM divides T_{drain} by F to account for other users who might be using the wireless channel. To illustrate, if three stations are simultaneously contending for the channel, each of them will roughly get $1/3$ of the time to transmit. Hence, the time to drain the queue is approximately $3\times$ higher compared to the case where only a single node is transmitting.

If T_{drain} exceeds $limit$ for *measurement intervals*, then this means that the network is suffering from bufferbloat. As a result, WQM decreases the buffer size and hence limits the latency. On the other hand, if T_{drain} is lower than $limit$ for *measurement intervals*, then WQM will increase the buffer size. To account for temporary bursty traffic, WQM observes the network measurements for two consecutive cycles before taking an action. This corrective action cannot adjust the buffer size beyond the acceptable limits, (*i.e.,* $B_{max} > B > B_{min}$ as described in Sec. IV-B below).

Finally, we would like to note that a larger $B_{initial}$ may be needed in multi-hop networks in order to achieve the best utilization. In reality, WQM starts with a sub-optimal buffer size as it prefers low latency. As shown in our experimental analysis (Sec. V), WQM handles both long-lived and short-lived flows in an efficient manner.

B. WQM ANALYSIS

Now, it is the time to find the upper bound (B_{max}) and the lower bound (B_{min}) of the buffer size. We are also going to estimate the amount of tolerated latency ($limit$). To find an upper bound on the buffer size, B_{max}, let us consider a single stream WiFi network (IEEE 802.11n). Also, let us fix the upper limit of the transmission rate to λ packets[1]/s. When the TCP stream happen to be in the congestion avoidance phase, the congestion window will grow upto W_{max}. At this moment in time, a packet is going to lost and the transmitter divides the TCP congestion window by two. The transmitter will move to the transmit phase again only after $\frac{W_{max}/2}{\lambda}$. The time to drain the buffer is calculated as B/λ s. In order to maintain full link utilization, the transmitter should always transmit at least a single frame just before the buffer gets totally empty (*i.e.,* $\frac{W_{max}/2}{\lambda} \leq B/\lambda$), or

$$B \geq \frac{W_{max}}{2} \tag{3}$$

Ideally, the transmission rate of the sender (*i.e.,* cwnd/*ARTT*) should be at least λ in order to make sure that the link is fully utilized. Hence, $\frac{W_{max}/2}{ARTT} \geq \lambda$, or,

$$\frac{W_{max}}{2} \geq ARTT \cdot \lambda \tag{4}$$

By combining (3) and (4), we get

$$B \geq \lambda \cdot ARTT \tag{5}$$

As a result, B_{max} is calculated using the BDP with the highest possible transmission rate and the estimated RTT. Basically, *ARTT* represents the transmission delay as propagation delay is considered negligible in wireless networks. Fig. 3 illustrates how to estimate the MAC overhead of transmitting a single A-MPDU over WiFI (IEEE 802.11n) network. As shown in the figure, *ARTT* is nothing but the sum of the transmission duration of the TCP segment T_{d-DATA} and the transmission duration of the TCP ACK T_{d-ACK}. This could be estimated using the following equations:

$$\begin{aligned} T_{d-DATA} &= T_{BO} + T_{DIFS} + 2 * T_{PHY} + T_{SIFS} + T_{BACK} \\ &\quad + K * (T_{MAC} + T_{DATA}) \qquad (6) \\ T_{d-ACK} &= T_{BO} + T_{DIFS} + 2 * T_{PHY} + T_{SIFS} + T_{BACK} \\ &\quad + K/2 * (T_{MAC} + T_{TCP-ACK}) \qquad (7) \end{aligned}$$

The system parameters are listed and described in Table 1. T_{BO} is the backoff interval in case the channel is found to be busy. Based on both MAC contention window and slot duration T_{slot}, the average backoff time is calculated as $\overline{T_{BO}} = (CW_{min} - 1) * T_{slot}/2$. The shortest inter-frame space is T_{SIFS} and the distributed inter-frame space is T_{DIFS}. T_{DATA} and T_{MAC} are aggregate frame and MAC header transmission time respectively. As defined by the standard specifications, just one A-MPDU may have K TCP segments, each of which has its own MAC header. In fact, these subframes can be as big as 64kB. Also, the maximum number of subframes per A-MPDU is 64. As a result,we add a transmission duration of $K * (T_{DATA} + T_{MAC})$ per A-MPDU. The TCP ACK transmission time is $T_{TCP-ACK}$ whereas T_{BACK} is the time to transmit a MAC-level block ACK frame. Moreover, we can

[1]Packets are used for illustration purposes.

TABLE 1. System parameters of WiFi Network (IEEE 802.11n) [9].

Parameter	Value
T_{slot}	slot duration = 9 μs
T_{SIFS}	shortest inter-frame space = 16 μs
T_{DIFS}	distributed inter-frame space = 34 μs
T_{PHY}	physical layer duration = 33 μs
CW_{min}	minimum size of contention window = 15
CW_{max}	maximum size of contention window = 1023
T_{BO}	average back-off interval = $(CW_{min}-1)*T_{slot}/2$
R	physical rate (Mb/s)
R_{basic}	basic physical rate = 6 Mb/s
K	maximum A-MPDU size
T_{MAC}	MAC header duration = L_{MAC}/R
L_{MAC}	MAC overhead = 38 B = 304 b
T_{DATA}	data frame duration = L_{DATA}/R
L_{DATA}	data frame size = 1500 B = 12000 b
$T_{TCP-ACK}$	TCP ACK duration = $L_{TCP-ACK}/R$
$L_{TCP-ACK}$	TCP ACK size = 40 B = 320 b
T_{BACK}	block ACK frame duration = L_{BACK}/R_{basic}
L_{BACK}	block ACK frame size = 30 B = 240 b

assume that TCP delayed ACK is used, and hence only $K/2$ frames are acknowledged. T_{PHY} is the transmission duration of both PHY preamble and header. We need the biggest buffer when the sender is transmitting with the highest possible Tx rate, 600 Mb/s in our case, and all A-MPDU are equipped with 64 subframes. The delay for transmitting an A-MPDU in this condition and its block ACK (per the exchange shown in Fig. 3) over a single hop is about 1.9 ms. According to (5), the upper bound on the buffer size B_{max} should be around 95 packets. B_{min} must be set to the maximum A-MPDU length allowed by the network. This is to avoid sending shorter aggregates which in turn limits the network capacity.

Next, we will calculate a lower bound on the queueing delay (*limit*). As shown earlier, the maximum allowable aggregate length varies with the link transmission rate. For example, a wireless channel that suffers from high interference might going to use the lowest possible link transmission rate (6.5 Mb/s for IEEE 802.11n radios). As per the ath9k [7] aggregation implementation logic, A-MPDU aggregation is disabled when transmitting at 6.5 Mb/s. As a result, *limit* must be at least equal to the transmission time of one frame using the lowest possible transmission rate. From (6) and (7), *limit* must be at least equal to 2.5 ms. Although different sessions may require different value of *limit*, we prefer to fix the value of *limit* in order to enhance the practicality of our algorithm. In fact, we test the value of *limit* $= 2.5$ ms in our experiments over various scenarios and concluded that 2.5 ms results in huge reduction in latency while maintaining the network throughput across a wide range of bandwidths, RTTs, and traffic loads.

V. EXPERIMENTAL ANALYSIS

A. IMPLEMENTATION DETAILS

In Linux, as well as other modern operating systems, buffers exist at multiple layers along the packet transmit path. The last buffer in Linux buffering architecture is the device internal buffer. This buffer is composed of a ring of descriptors. Each one of these descriptors points to the location of a single packet in main memory. Between the device internal buffer and the IP stack, there exist another type of buffer called the transmit queue (*txqueue*). Unlike the device buffer that treats all packets equally, queueing disciplines (also known as qdiscs) are used to schedule packets in the transmit queue, providing a mean to treat packets differently. Moreover, the device buffer is sized by the number of descriptors which are nothing but pointers. Hence, it is very difficult to accurately estimate the time required to empty this buffer. As a result, we decide to keep the default value of the device buffer and tune *txqueue* length instead. By default, *txqueue* is set to 1000 packets in current Linux distributions to support networks with high BDP. This buffer should be made smaller in slower networks to avoid high latency. WQM controls the Transmit Queue length *(txqueuelen)* using the *ifconfig* utility and gathers channel related information using the *iw* utility.

WQM is implemented as a daemon running in Linux user space. The source code is available at [10]. WQM passively probes the channel to find out the healthiness of the wireless channel. In fact, WQM is synchronized with the default rate control algorithm in Linux that is called Minstrel [6], because the link transmission rate is guaranteed to be fixed over this interval. Periodically, WQM obtains important statistics such as the link rate, the number of packets in the buffer, the aggregate length, and the percentage of time the channel was busy in the last look-around. In reality, this look-around interval is found to be sufficient to collect meaningful samples without wasting too many CPU cycles. Further, our experimental analysis in Section V shows that this interval is sufficient to respond quickly to changes in the environment.

As explained earlier, if WQM estimates the queueing delay to be longer than the desired target, then it is going to reduce the buffer size to lower latency. In reality, the buffer size should be at least equal to the number of subframes in an A-MPDU. Otherwise, the buffer will be limiting the network utilization without improving latency. Alternatively, if the time to drain the queue is less than the predefined target, WQM can safely increase the buffer size. Our implementation of WQM increases the buffer size by one packet in response to measurements outlined in Algorithm 1. When required by the algorithm, WQM decreases the buffer size by half to make sure the network latency remains within the desired range. Thus, WQM strictly prefers low latency over high goodput. Fig. 4 illustrates this behavior by showing the variation of buffer size over time. This Additive Increase Multiplicative Decrease (AIMD) behavior of the algorithm is chosen as it experimentally outperforms all other possible alternatives, namely AIAD, MIAD, and MIMD in terms of end-to-end delay as shown in Fig. 5. The experiment setup is detailed in the following section.

WQM performance is compared to the default buffering scheme in Linux, where the buffer size is fixed to 1000 packets, and two state of the art bufferbloat solution in the literature, namely CoDel [3] and PIE [4]. Both CoDel and PIE are considered to be parameterless with auto-tuning functionality. Hence, we do not need to specify any additional parameters for them.

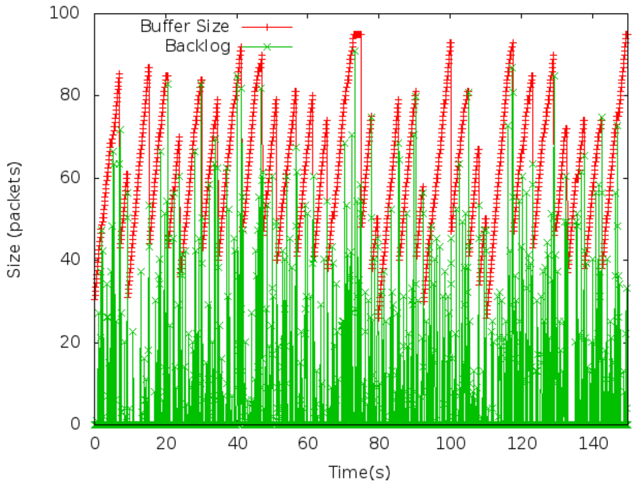

FIGURE 4. WQM buffer size adaptation in response to variation in queue occupancy. This figure represents the AIMD behavior of WQM.

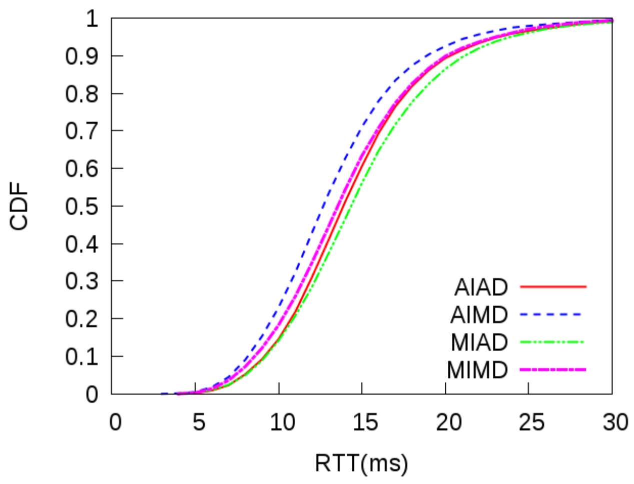

FIGURE 5. Comparing the round trip delay of various versions of WQM algorithm.

B. EXPERIMENTAL SETUP

WQM is evaluated over a wireless testbed on our campus. Our testbed consists of 10 Shuttle [11] computers equipped with Intel Core 2 Duo processors and 1 GB of RAM. Each computer has an IEEE 802.11n wireless network interface card (TP-Link WDN4800/Atheros AR9380), which operates in dual-band and supports upto 3-streams simultaneously, supporting link rates upto 450 Mb/s. The testbed node locations are shown in Fig. 6. Unless otherwise stated, the nodes are placed around 10 m apart from each other. We repeat the experiments multiple times over various source and destination pairs across the testbed to offset any location-specific behavior. We chooses to use the quite band (5 GHz U-NII radio band) to avoid collisions with our university WiFi network. We patch the recent stable Linux kernel (3.17.7) with

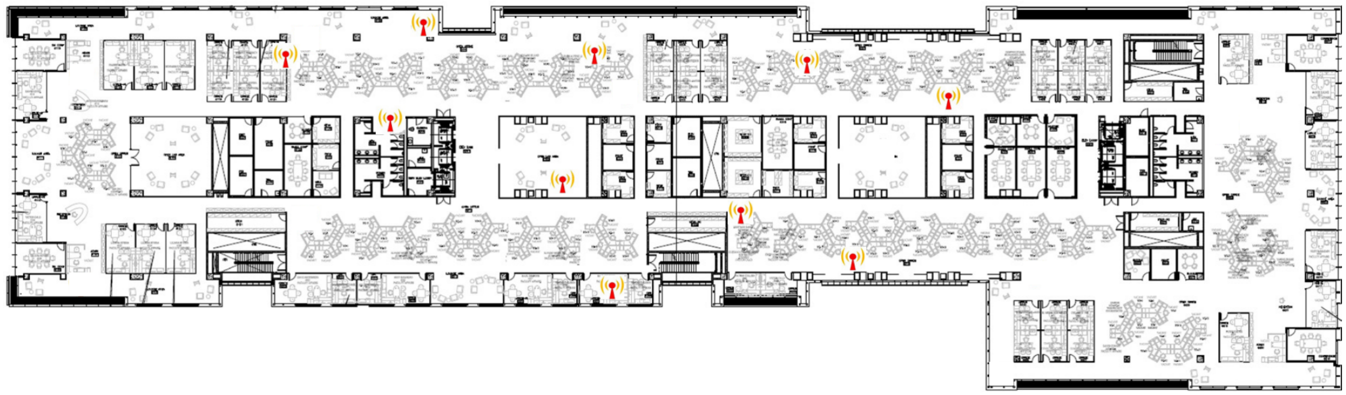

FIGURE 6. The floor plan showing testbed node locations (identified by a radio icon).

web10g [12] to monitor various TCP statistics. The device driver we use is ath9k [7] with Minstrel [6] as the rate control algorithm. Also, we use TCP Cubic, the default TCP version on our stock Linux. To simulate a large file transfer, we run netperf [13] for 100 seconds.

C. EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of WQM experimentally. Our experiments could be classified into two main groups based on the number of concurrent flows used in the experiments, namely single flow scenarios and multi-flow scenarios.

1) SINGLE FLOW SCENARIOS

In this set of experiments, we measure the goodput and latency of a single long lived flow. We compare WQM performance to two AQM based solutions as well as the default static buffer size. We start by evaluating WQM over scenarios with multiple hops. In this experiments, we simply increase the number of hops between the sender and the receiver gradually from one to three. Hence, packets are queued multiple times before reaching their destinations. The cumulative distribution function (CDF) of the latency over various topologies and the corresponding average goodput are shown in Fig. 7 and Fig. 8, respectively. We would like to note that unless otherwise stated, all the results in this section are averaged over at least three runs and the error bars represent the minimum and maximum values. For various scenarios, WQM manages to lower the round trip time by at least 5× compared to both PIE and the static buffer sizing scheme and 2× compared to CoDel. To give an example, WQM manages to reduce the round trip time in the three hops scenario from 261.3 ms using PIE to only 49.47 ms at the cost of at most 5% in goodput reduction. We attribute the ability of WQM to outperform other schemes in controlling the queuing delay to the achieved level of frame aggregation. Furthermore, it is clear that the required buffer size in wireless networks is much lower than the buffer size limit in PIE which is 1000 packets.

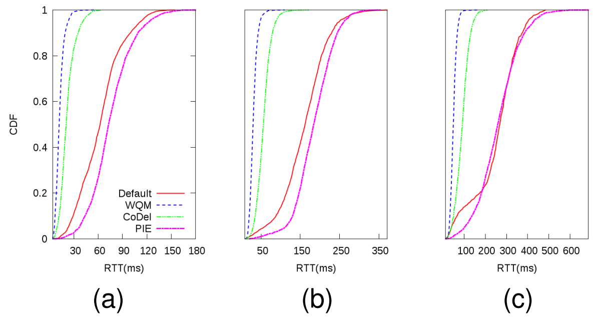

FIGURE 7. Single flow latency for various hop count. (a) One hop. (b) Two hops. (c) Three hops.

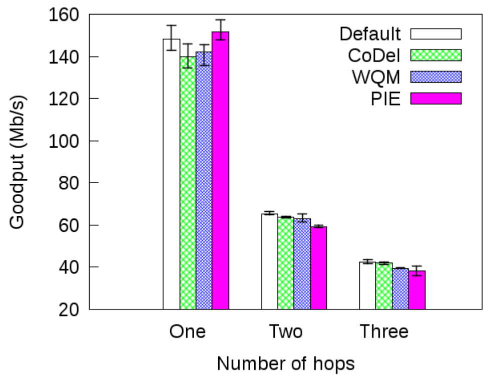

FIGURE 8. Single flow goodput for various hop count.

In order to support backward compatibility, IEEE 802.11n devices disable frame aggregation if the receiver is not capable of dealing with aggregates. To test this scenario, we repeat the previous set of experiments after disabling A-MPDU aggregation. For various hop counts, the RTT CDF and average goodput are shown in Fig. 9 and 10 respectively. Compared to the default case with 1000 packets buffer, WQM achieves upto 7× reduction in latency while having similar goodput. As expected, WQM performs as well as CoDel and PIE in terms of delay and goodput when aggregation is disabled. This set of experiments show that even if the

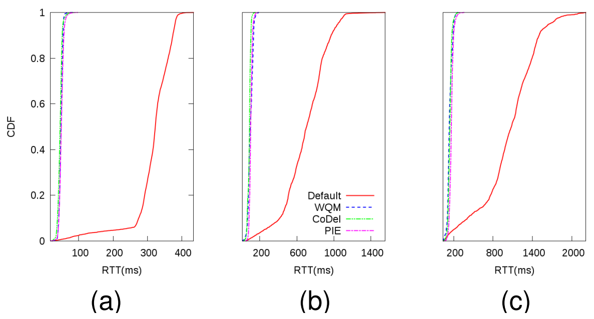

FIGURE 9. Single flow latency for various hop count after disabling A-MPDU frame aggregation. (a) One hop. (b) Two hops. (c) Three hops.

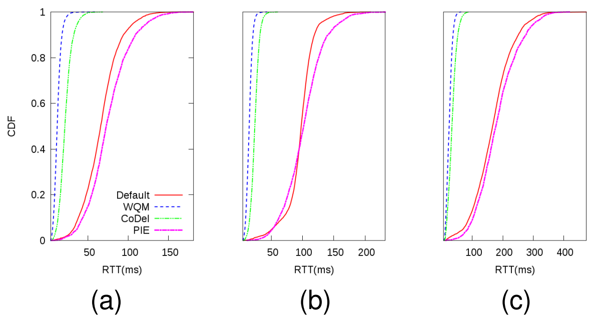

FIGURE 11. Single flow latency while varying testbed size. (a) 10m. (b) 20m. (c) 30m.

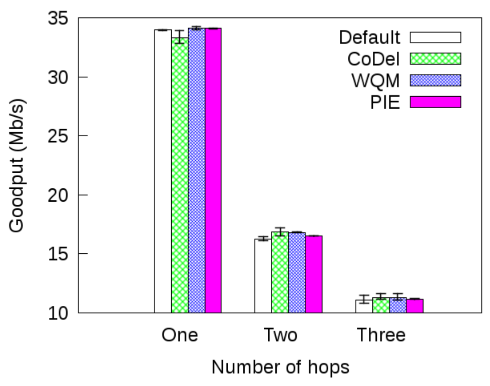

FIGURE 10. Single flow goodput for various hop count after disabling A-MPDU frame aggregation.

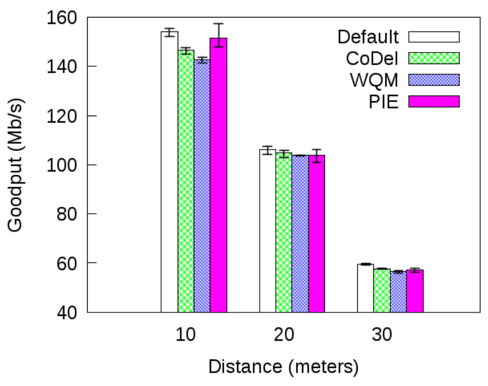

FIGURE 12. Average goodput achieved while varying the distance between nodes in the testbed.

Wi-Fi devices are not deployed in green field mode, *i.e.,* the network is not solely composed of IEEE 802.11n devices, WQM can still maintain an acceptable network latency. It is also worth noting that these experiments show that A-MPDU frame aggregation helps reducing the latency from about 2 sec to only 500 ms in the worst case. This is attributed to the ability of the aggregation scheme to drain the buffer quickly.

To evaluate WQM under various channel conditions, the same set of experiment are repeated three times while varying the distance between the testbed nodes. We start with the default distance in our testbed which is 10 m, then increase it to 20 m, and finally to 30 m. Delay and goodput results are shown in Fig 11 and 12 respectively. As expected, nodes that are far from each other suffer from longer delay compared to closer nodes. In all the three cases, WQM outperforms CoDel, PIE and the static buffer sizing scheme in terms of latency. This reduction comes with at most 7% drop in goodput. It is important to highlight that the gap in goodput between WQM and the other schemes shrinks as the distance between the nodes gets longer. This happens because the rate control algorithm, which is enabled by default in our testbed, reduces the transmission rate when the transmitter and the receiver are far from each other in order to increase the link reliability. This in turn lowers the BDP *i.e.,* the required buffer in the network. As a result, the effect of WQM small buffer on network utilization is going to be minimal in this case.

Finally, we evaluate the performance of WQM using short flows. In all previous experiments, long flows of 100 s are used to simulate large file transfers that are able to saturate the channel and fill the buffers. However, short flows are also common in real life. To test this scenario we repeat the same set of experiments while varying the flow duration from 5, 10, to 15 s and observe the delay and goodput. To get accurate results, every experiment is repeated around 10 times and the average is reported. The round trip time CDF is shown in Fig. 13 while the goodput CDF is shown in Fig. 14. It can be observed that in all cases the delay is bounded to 150 ms which is a direct implication of not building the buffers. However, WQM is still achieving the best reduction in RTT compared to all other schemes. This reduction comes at the cost of at most 8% drop in goodput.

2) MULTI FLOW SCENARIOS

In this section, we evaluate the performance of WQM while running multiple concurrent flows instead of only one between the transmitter and the receiver. In the first set of experiments, the number of concurrent flows is increased from one to five flows. For every scenario, we measure the RTT and goodput for WQM, CoDel, PIE as well as the static buffering scheme. The RTT CDF for 1, 3, and 5 flows are shown in Fig. 15 and the average per flow goodput for the three cases are shown in Fig. 16. Our results show that

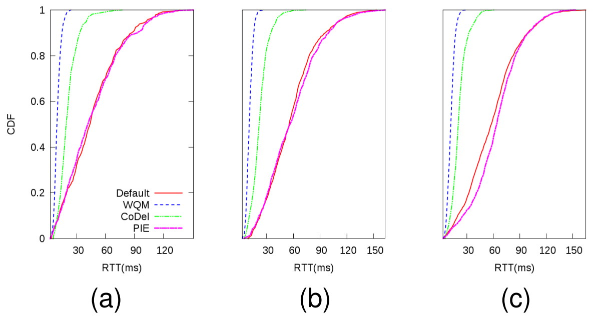

FIGURE 13. Latency while varying the flow duration. (a) 5s. (b) 10s. (c) 15s.

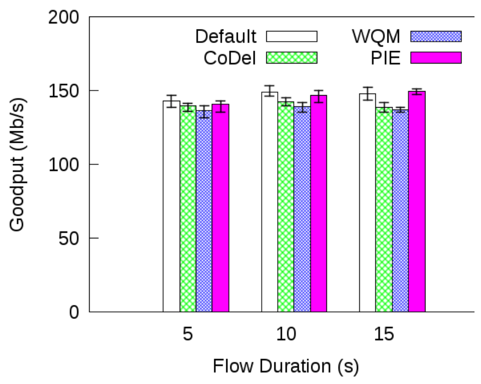

FIGURE 14. Average goodput achieved while varying the flow duration.

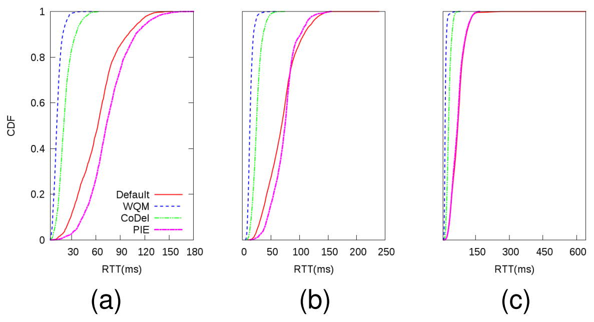

FIGURE 15. Latency of various concurrent flows over a single hop topology. (a) One flow. (b) Three flows. (c) Five flows.

WQM lowers the latency by at least 5× for all scenarios when compared to both PIE and the static buffering scheme. This reduction in latency comes at the cost of at most 13% less goodput. when compared to CoDel, WQM almost halves the latency while preserving goodput. As highlighted in the literature [14], drop-tail buffering might cause severe throughput unfairness between flows. When the number of flows is 3, the JFI (Jains Fairness Index) value for the static buffering scheme is 0.7, compared to 0.99 for WQM. The reason for this anomaly is the large buffers used in the static buffering scheme which lead to severe unfairness between flows as a single flow may fill up the buffer and starve others. WQM avoids this problem by adaptively sizing the buffer.

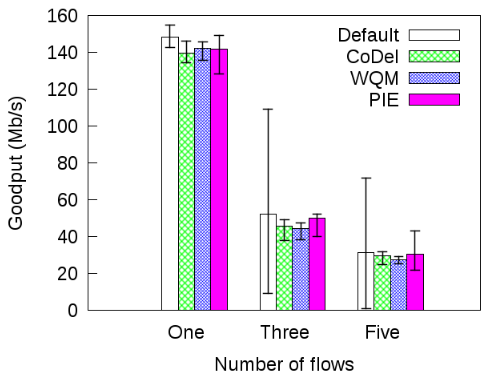

FIGURE 16. Average goodput with multiple flows over a single hop topology.

As mentioned earlier, frame aggregation might be disabled in certain cases. To test the performance of WQM under these scenarios, we repeat the previous set of experiments while disabling A-MPDU aggregation. The latency comparison is highlighted in Fig. 17 and the average per flow goodput is demonstrated in Fig. 18. When compared to the default static buffer size, WQM reduces the delay by around 7× while getting similar goodput. WQM is performing as well as both CoDel and PIE in this case. This is happening because

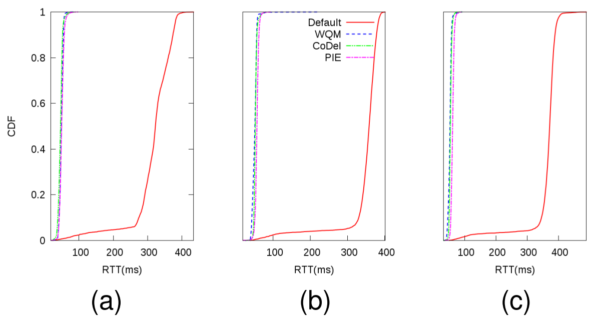

FIGURE 17. Latency of multiple concurrent flows over a single hop topology without A-MPDU aggregation. (a) One flow. (b) Three flows. (c) Five flows.

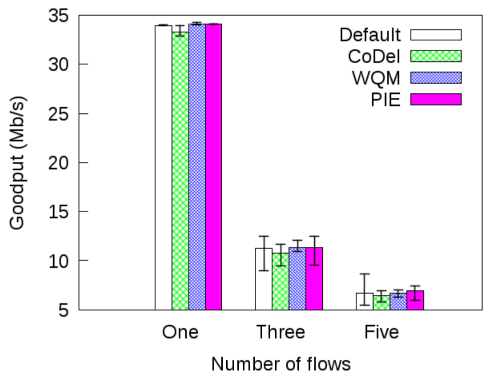

FIGURE 18. Average goodput with multiple flows over a single hop topology without A-MPDU aggregation.

WQM accounts for frame aggregation when selecting the optimal buffer size. As illustrated in Sec. IV-B, the number of sub-frames per aggregate, which is one packet in this case, will be used as a lower bound to buffer size in WQM. In fact, this one packet limit is what CoDel and PIE use in their buffers. After all, this set of experiments show that WQM still performs as well as the state of the art even at border cases.

Last but not least, we evaluate WQM over both multi-hop and multi-flow scenarios. In these set of experiments, we organize the testbed nodes to be in a parking lot topology, as illustrated in Fig. 19. In this topology, three flows traverse the testbed such that the 1st flow travels upto one hop away from the source, the 2nd flow travels upto two hops and the 3rd flow is the only one that reaches the third hop. We repeat these experiments several times with and without activating the rate control algorithm. When disabled, the link rates in the testbed are manually fixed to one of the following rate: 6.5, 13, 65 or 144.4 Mb/s. Latency per flow and the aggregate goodput are displayed in Fig. 20. Similar to the previous experiments, we average the results over at least three runs. Error bars in the figure represent our maximum and minimum results. When comparing WQM to the static buffering methodology, we ind out that the former lowers the latency by 8× for the single hop flow, 6× for the two hops flow and more than 4× for the three hops flow. This latency reduction does not come with a significant goodput reduction. This proofs that static large buffers are not mandatory in wireless networks. similarly, WQM defeats CoDel in all scenarios. For example, WQM halves latency when Minstrel is enabled while improving goodput. Since aggregation is disabled at 6.5 Mb/s link rate, PIE performance is close to WQM performance. As the rate increases, the gap between PIE and WQM gets bigger as longer aggregates are being transmitted. In the worst case, PIE suffers from 7× more latency compared to WQM.

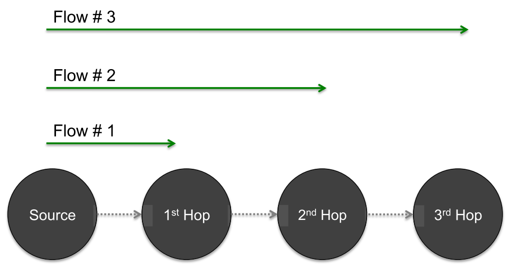

FIGURE 19. Illustration of the parking lot topology used in our experiments.

VI. RELATED WORK

Allocating the optimal buffer size in networks is an important problem. For core Internet routers, the convention is to have the buffer sized according to the BDP [8], where the buffer need to be at least the product of the packet round trip delay and the capacity of the bottleneck link across the route. The main goal of this method is to keep the bottleneck link fully utilized even when the TCP sender is operating in the congestion avoidance phase. In the case of multiple flows that are desynchronized, the needed buffer to achieve a full link utilization is much less than the BDP [15] [16]. In fact, this method cannot be applied directly to wireless networks because in this type of networks the packet transmission delay is linked with the available bandwidth [17]. To illustrate, a wireless node must contend with other devices in the neighborhood before getting a transmission opportunity which is not the case in wires networks. Moreover, every sent frame need to be acknowledged before proceeding to send a new frame whereas a wired device can send multiple frames back to back. It is well known that the transmission delay in wired networks is negligible when compared to propagation delay. On the other hand, what matters the most in wireless networks is the time to push your frame into the channel. This coupling between delay and bandwidth in wireless links makes the buffer sizing problem more challenging.

Only few papers tackled the problem of buffer sizing in the wireless domain. A* is an algorithm to fine tune the Access Point (AP) buffer dynamically [18]. In order to find the optimal buffer size, A* keeps an eye on packet service rate. It alters the buffer size based on buffer occupancy. One of the main limitations of A* is the fact it could not be easily extended to support multi-hop scenarios (*i.e.,* it can deal only with AP buffer). Furthermore, A* adds a timestamp for each and every packets coming from the network. This is clearly an unnecessary overhead. Lastly, A* performance evaluation was done using IEEE 802.11g radios only; and hence this scheme might not be able to cope up with new enhancements in the WiFi standard such as frame aggregation. In fact, an experimental study evaluated A* over various scenarios and proved that it performs sub-optimally in practice [19]. Recently, Høiland-Jørgensen *et al.* [20] also tackled bufferbloat in APs with the goal to reduce latency and improve airtime fairness. Their proposal was built on top of FQ-CoDel which is one of the queuing disciplines in Linux. We believe that the mesh environment imposes further challenges that are not addressed in this work.

DNB or Distributed Neighborhood Buffer [5] tried to solve the problem of optimal buffer sizing in mesh networks. In this scheme, the network wide buffer size is calculated and then distributed among a set of competing nodes based on a certain cost function with the goal of saturating the spectral resources available. The authors concluded that sizing buffers collectively in WMNs nodes lead to small buffers, in the range of 1-3 packets. In fact, this approach suffer from multiple limitations. First, it does not adapt to changes in link capacities; instead, it assumes that links have static rates and based on this assumption it finds optimal buffer size for every mesh node in the network. Second, it is obvious that these extremely small buffers are going to limit the level of frame aggregation and hence degrade network utilization. Finally, the proposed scheme was optimized to be used with single TCP flow. As a result, good performance with multi flow scenarios is not guaranteed. Also for mobile ad-hoc networks, Dousse [21] proposed the reduction of

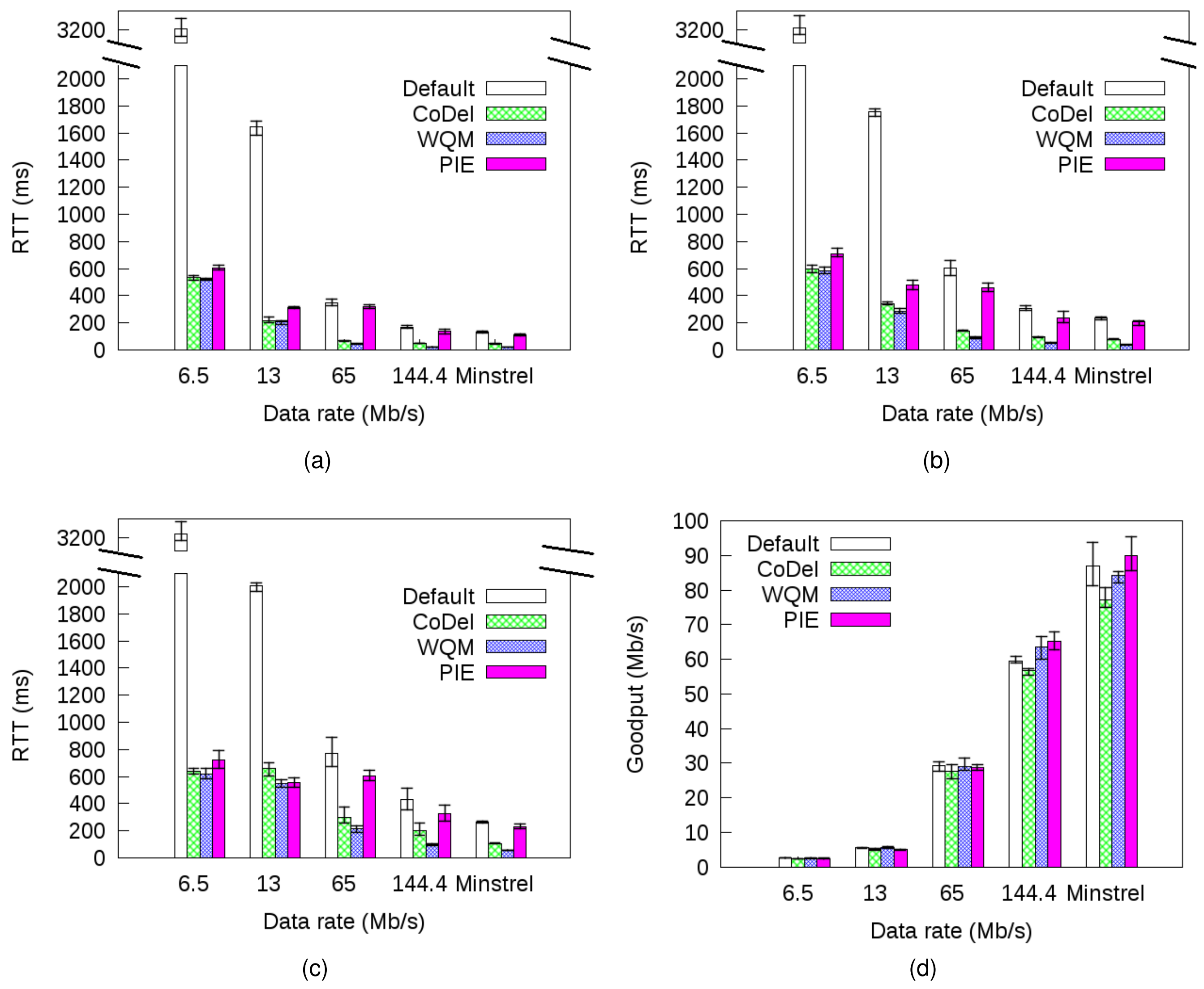

FIGURE 20. **Goodput and latency per flow over the parking lot topology. (a) One hop flow. (b) Two hops flow. (c) Three hops flow. (d) Total goodput.**

buffer size on relay nodes to only one packet to mitigate the problem of low throughput in multi-hop networks. Obviously, this is infeasible for Wi-Fi networks with A-MPDU size of tens of subframes.

Recently, several papers empirically evaluated the bufferbloat phenomena in both wired and wireless networks [22]–[24]. In our previous work [25], we proposed an initial attempt to solve bufferbloat in the wireless domain. Our preliminary results showed a significant reduction in end-to-end delay compared to static buffers. However, this work lacks an extensive evaluation. For instance, the comparison to the state of the art bufferbloat solution was not done. Moreover, the performance of the scheme over networks that do not support frame aggregation was not evaluated. Furthermore, experimenting short flows was also lacking. Finally, the selection of the detailed parameters that affect the dynamics of the algorithm was not justified. Furthermore, the problem of bufferbloat has been tackled in heterogeneous networks [26] and cellular networks [27]–[29]. As an example in the cellular domain, Jiang *et al.* proposed a scheme called Dynamic Receive Window Adjustment (DRWA) [30] that modifies the TCP stack instead of dealing with the buffer size directly. DRWA limits the amount of buffering inside the network by tuning the size of TCP receive window. Also for cellular networks, Chan *et al.* proposed a scheme called SoD or (Sum-of-Delay) [31] that estimates the optimal buffer size in 3G/4G networks. The main idea is to modify TCP congestion control to function based on estimated queue length instead of packet loss events. Both of these schemes have practical limitations because of the large deployed base of TCP.

Warrier *et al.* [32] proposed a differential backlog congestion control for wireless networks. The idea is to throttle the flow control of the sender based on the queue back pressure. Hence, if the queue grows beyond certain threshold, the sender will be reducing its transmit rate and vice versa. This is opposite to what WQM does. One limitation of such approach is the need of the queue occupancy information from several hops in the multi-hop networks. Transferring this information over multiple hops is a considerable overhead that will affect the practicality of the proposed approach. Alternatively, WQM works with local knowledge even for multihop networks. Recently, Byeon *et al.* [33] proposed a scheme to adaptively change the aggregate size based on nodes mobility patters. We envision that such scheme would be complementary to WQM.

VII. CONCLUSION

Recent enhancements in IEEE 802.11n/ac standard specifications, such as A-MPDU frame aggregation, make it more difficult to find the optimal buffer size in WiFi based networks. Large buffers may lead to long end-to-end delays in the order of seconds. We are proposing a novel, practical, and dynamic buffer management tool called WQM. It selects the optimal buffer size based on several parameters such as network congestion, channel interference intensity, and the level of frame aggregation. WQM is evaluated using a 10 nodes wireless testbed. We prove through experiments over various single-hop and multi-hop scenarios that WQM is able to cut down the queuing delay by a factor of 8× when compared to static buffer sizing. Further, WQM outperforms other state of the art bufferbloat solutions such as CoDel and PIE by a factor of 7× in terms of delay reduction. In the worst case, this reduction comes at the cost of 8% drop in goodput. Finally, we show that WQM improves flow fairness by preventing a single flow from saturating the buffers.

As future work, we are looking into several new challenges. One of them is to evaluate WQM over various types of flows across different topologies. Furthermore, we would like to assess WQM using selective drop mechanisms. Also, it is interesting to evaluate how does TCP pacing interacts with frame aggregation in wireless networks. Moreover, we would like to understand the consequences of introducing an adaptive look-around interval to WQM. Finally, we would also like to test WQM using wireless devices with IEEE 802.11ac compatible radios.

REFERENCES

[1] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the Internet," *Commun. ACM*, vol. 55, pp. 57–65, Jan. 2012.
[2] C. Staff, "Bufferbloat: What's wrong with the Internet?" *Commun. ACM*, vol. 55, no. 2, pp. 40–47, Feb. 2012.
[3] K. Nichols and V. Jacobson, "Controlling queue delay," *Queue*, vol. 10, no. 5, pp. 1–15, May 2012.
[4] R. Pan, P. Natarajan, F. Baker, and G. White, *Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem*, document RFC 8033, 2017.
[5] K. Jamshaid, B. Shihada, A. Showail, and P. Levis, "Deflating link buffers in a wireless mesh network," *Ad Hoc Netw.*, vol. 16, pp. 266–280, May 2014.
[6] D. Smithies and F. Fietkau. *Minstrel: Rate Control Algorithm for mac80211*. Accessed: May 15, 2018. [Online]. Available: http://wireless.kernel.org/en/developers/Documentation/mac80211/RateControl/minstrel
[7] *Ath9k: Atheros Linux Wireless Drivers*. Accessed: May 15, 2018. [Online]. Available: http://wireless.kernel.org/en/users/Drivers/ath9k
[8] C. Villamizar and C. Song, "High performance TCP in ANSNET," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 5, pp. 45–60, Oct. 1994.
[9] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, Standard IEEE 802.11, IEEE LAN/MAN Standards Committee, 2012.
[10] *Wireless Queue Management (WQM) Source Code*. Accessed: May 15, 2018. [Online]. Available: https://www.shihada.com/system/files/2016-10/WQM_0.zip
[11] *Shuttle Cube Computers*. Accessed: May 15, 2018. [Online]. Available: http://www.shuttle.com
[12] *The Web10g Project*. Accessed: May 15, 2018. [Online]. Available: http://www.web10g.org/
[13] *Netperf Benchmarking Tool*. Accessed: May 15, 2018. [Online]. Available: http://www.netperf.org/netperf/
[14] S. R. Pokhrel, M. Panda, and H. L. Vu, "Analytical modeling of multipath TCP over last-mile wireless," *IEEE/ACM Trans. Netw.*, vol. 25, no. 3, pp. 1876–1891, Jun. 2017.
[15] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," in *Proc. Conf. Appl., Technol., Architectures, Protocols Comput. Commun. (SIGCOMM)*, 2004, pp. 281–292.
[16] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, and T. Roughgarden, "Routers with very small buffers," in *Proc. IEEE INFOCOM*, Apr. 2006, pp. 1–11.
[17] K. Chen, Y. Xue, S. H. Shah, and K. Nahrstedt, "Understanding bandwidth-delay product in mobile ad hoc networks," *Comput. Commun.*, vol. 27, no. 10, pp. 923–934, Jun. 2004.
[18] T. Li, D. Leith, and D. Malone, "Buffer sizing for 802.11-based networks," *IEEE/ACM Trans. Netw.*, vol. 19, no. 1, pp. 156–169, Feb. 2011.
[19] D. Taht. *What I Think is Wrong With eBDP in Debloat-Testing*. Accessed: May 15, 2018. [Online]. Available: https://lists.bufferbloat.net/pipermail/bloat-devel/2011-November/000280.html
[20] T. Høiland-Jørgensen, M. Kazior, D. Täht, P. Hurtig, and A. Brunstrom, "Ending the anomaly: Achieving low latency and airtime fairness in WiFi," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, Santa Clara, CA, USA, Feb. 2017, pp. 139–151.
[21] O. Dousse, "Revising buffering in CSMA/CA wireless multihop networks," in *Proc. IEEE SECON*, Jun. 2007, pp. 2356–2360.
[22] M. Allman, "Comments on bufferbloat," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 1, pp. 30–37, Jan. 2012.
[23] A. Showail, K. Jamshaid, and B. Shihada, "An empirical evaluation of bufferbloat in IEEE 802.11 n wireless networks," in *Proc. Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2014, pp. 3088–3093.
[24] O. Hohlfeld, E. Pujol, F. Ciucu, A. Feldmann, and P. Barford, "A QoE perspective on sizing network buffers," in *Proc. Conf. Internet Meas. Conf. (IMC)*, 2014, pp. 333–346.
[25] A. Showail, K. Jamshaid, and B. Shihada, "WQM: An aggregation-aware queue management scheme for IEEE 802.11 n based networks," in *Proc. ACM SIGCOMM Workshop Capacity Sharing Workshop (CSWS)*, 2014, pp. 15–22.
[26] B. Felix, A. Santos, B. Kantarci, and M. Nogueira, "CD-ASM: A new queuing paradigm to overcome bufferbloat effects in HetNets," in *Proc. IEEE 28th Annu. Int. Symp. Pers., Indoor, Mobile Radio Commun. (PIMRC)*, Oct. 2017, pp. 1–6.
[27] Y. Guo, F. Qian, Q. A. Chen, Z. M. Mao, and S. Sen, "Understanding on-device bufferbloat for cellular upload," in *Proc. ACM Internet Meas. Conf.*, 2016, pp. 303–317.
[28] X. Liu, F. Ren, R. Shu, T. Zhang, and T. Dai, "Mitigating bufferbloat with receiver-based TCP flow control mechanism in cellular networks," in *Advances in Computer Communications and Networks: From Green, Mobile, Pervasive Networking to Big Data Computing* (River Publishers Series in Communications), K. Sha, A. Striegel, and M. Song, Eds. Dec. 2016, p. 65.
[29] J. D. Beshay, A. T. Nasrabadi, R. Prakash, and A. Francini, "On active queue management in cellular networks," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, May 2017, pp. 384–389.
[30] H. Jiang, Y. Wang, K. Lee, and I. Rhee, "DRWA: A receiver-centric solution to bufferbloat in cellular networks," *IEEE Trans. Mobile Comput.*, vol. 15, no. 11, pp. 2719–2734, Nov. 2016.
[31] S. C. F. Chan, K. M. Chan, K. Liu, and J. Y. B. Lee, "On queue length and link buffer size estimation in 3G/4G mobile data networks," *IEEE Trans. Mobile Comput.*, vol. 13, no. 6, pp. 1298–1311, Jun. 2014.
[32] A. Warrier, S. Janakiraman, S. Ha, and I. Rhee, "DiffQ: Practical differential backlog congestion control for wireless networks," in *Proc. INFOCOM*, Apr. 2009, pp. 262–270.
[33] S. Byeon, K. Yoon, O. Lee, S. Choi, W. Cho, and S. Oh, "MoFA: Mobility-aware frame aggregation in Wi-Fi," in *Proc. 10th ACM Conf. Emerg. Netw. Experim. Technol. (CoNEXT)*, 2014, pp. 41–52.

AHMAD SHOWAIL received the B.Sc. (Hons.), M.Sc., and Ph.D. degrees from the King Fahd University of Petroleum and Minerals in 2005. Before joining the King Abdullah University of Science and Technology (KAUST), he was a System Engineer at SABIC. He is currently an Assistant Professor of computer science with the College of Computer Science and Information Technology, University of Prince Mugrin. He published several papers in top journals and magazines and filed a U.S. patent. His research focuses on quality of service in wireless multi-hop networks. While at KAUST, he was a recipient of the Academic Excellence Award (top 5%).

BASEM SHIHADA (SM'12) was a Visiting Faculty with the Computer Science Department, Stanford University, in 2009. He is currently an Associate and a Founding Professor of computer science and electrical engineering from the Computer, Electrical and Mathematical Sciences and Engineering Division, King Abdullah University of Science and Technology. His current research covers a range of topics in energy and resource allocation in wired and wireless communication networks, including wireless mesh, wireless sensor, multimedia, and optical networks. He is also interested in SDNs, IoT, and cloud computing.