WQM: An Aggregation-aware Queue Management Scheme for IEEE 802.11n based Networks

Ahmad Showail, Kamran Jamshaid, and Basem Shihada
CEMSE Division
King Abdullah University of Science and Technology
Thuwal, Saudi Arabia
{ahmad.showail, kamran.jamshaid, basem.shihada}@kaust.edu.sa

ABSTRACT

Choosing the right buffer size in Wi-Fi networks is challenging due to the dynamic nature of the wireless environment. Over buffering or 'bufferbloat' may produce unacceptable end-to-end delays, while static small buffers may limit the performance gains that can be achieved with various 802.11n enhancements, such as frame aggregation. We propose WQM, a queue management scheme customized for wireless networks. WQM adapts the buffer size based on measured link characteristics and network load. Furthermore, it accounts for aggregate length when deciding about the optimal buffer size. We implement WQM on Linux and evaluate it on a wireless testbed. WQM reduces the end-to-end delay by up to $8\times$ compared to Linux default buffer size, and $2\times$ compared to CoDel, the state-of-the-art bufferbloat solution, while achieving comparable network goodput. Further, WQM improves fairness as it limits the ability of a single flow to saturate the buffer.

1. INTRODUCTION

Recent measurement studies have uncovered significant overbuffering in data networks. While big buffers may potentially increase throughput by limiting packet drops, large queues can result in high latency. This phenomenon of 'bufferbloat' [3] can create delays up to orders of seconds. With falling memory prices and the fallacy 'more is better', this performance degradation from large buffers can be observed in many networking devices, including end-user equipment such as DSL, cable routers or wireless Access Points (APs).

Addressing bufferbloat in wireless devices, such as APs, is a significant challenge. First, the wireless link capacity is not fixed. Wireless devices use a rate control algorithm to select their transmission rate based on several variables, such as the distance between the transmitter and the receiver and the channel noise floor. For example, IEEE 802.11n link rate can vary from 6.5 Mb/s to 600 Mb/s. With over two orders of magnitude variation in link rate, using a fixed buffer size can significantly degrade the network performance. Second, wireless is a shared medium and hence wireless link scheduling is not independent. Thus, a node's share of the wireless channel depends on other wireless nodes in the neighborhood that are also contending for channel access. As a result, the usable link capacity is typically much smaller than the physical link rate. This also affects the amount of buffering needed in the network. Finally, inter-service rate for a wireless packet transmission can vary, since packets may need to be retransmitted multiple times before they reach their destination.

802.11n/ac standard specifications introduced MAC-layer frame aggregation to improve network performance. Using Aggregate MAC Protocol Data Unit aggregation (A-MPDU), a wireless node can transmit up to 64 subframes in a single channel access. The aggregation logic is left open to the vendor's implementation. Always transmitting a maximum-sized A-MPDU may maximize throughput, but will increase delays if a node needs to wait to assemble 64 MPDUs from higher layers. In contrast, the current Linux implementation sends as many frames as currently available in the buffers, resulting in A-MPDUs with variable frame sizes. As shown in our previous work [8], A-MPDU frame aggregation helps deflating the buffer which results in significant queueing delay reduction. However, this reduction is not always enough to achieve acceptable Quality of Service (QoS).

To address these challenges, we designed and implemented WQM, an aggregation-aware queue management scheme for wireless networks. WQM identifies and distinguishes between 'good' and 'bad' buffers. Good buffers are buffers needed to absorb bursty traffic, while bad buffers only contribute to network latency without any noticeable improvement in throughput. WQM is both practical and incrementally deployable; it uses existing data traffic as probe for network measurements and does not incur any additional overhead. To account for channel variability, WQM periodically calculates the time needed to drain the buffer based on the current transmission rate. It then adjusts the buffer size to maintain the network QoS, reducing queueing delays where necessary, while allowing sufficient buffers to saturate available network capacity. Further, WQM incorporates MAC behavior to get accurate estimates of queue draining time. WQM is implemented in Linux and our experimental results show that it reduces end-to-end delay by up to 8× with a slight drop in network throughput when compared to the default queue size used in stock Linux. It also achieves lower latency when compared to the recently proposed queue management scheme called CoDel .

To the best of our knowledge, this is the first attempt to address the buffer sizing problem for 802.11n networks. Compared to other efforts in the wired and the wireless domain, WQM is the only scheme that accounts for frame aggregation when deciding about the optimal queue size.

2. RELATED WORK

There is limited work addressing the buffer sizing problem in the wireless space. Li *et al.* [6] propose A* algorithm that adaptively sets AP buffer size in single-hop wireless networks. A* uses the ratio between the max. acceptable queueing delay and packet mean service rate to find the optimal buffer size. This buffer size is further tuned by monitoring buffer occupancy. A* operates on AP buffers, and it is unclear if it can easily be extended to multi-hop wireless networks. Moreover, to calculate packet service time, A* attaches a timestamp to every packet entering the queue. This overhead affects the overall network performance. Finally, A* was not evaluated using real 802.11n devices, and hence it is unclear how it will react to various enhancements such as frame aggregation. In fact, some performance evaluation studies show that A* may result in sub-optimal performance when tested over more practical scenarios [10].

Distributed Neighborhood Buffer [5] targets the buffer sizing problem in Wireless Mesh Networks (WMNs). The authors first calculate a cumulative neighborhood buffer based on link interference constraints, and then distribute it among competing nodes using a cost function to ensure efficient spectral utilization. This method results in buffers as small as 1-3 packets at most mesh nodes. However, this approach has several limitations. First, it assumes fixed link rates and calculates the buffer sizes accordingly. Second, the scheme targets 802.11 a/b/g radios; using these small buffers in 802.11n may limit the level of frame aggregation. Finally, the scheme is optimized for single TCP flows, and its performance in multi-flow scenarios is not guaranteed.

Active Queue Management (AQM) techniques attempt to prevent large queue buildup at intermediary hosts through proactive, probabilistic packet drop. However, these algorithms failed to gain traction because of the complexity of correctly setting various configuration knobs. Recently, a no-knobs AQM technique called CoDel (Controlled Delay) [7] was proposed. Instead of measuring the queue size, CoDel tracks the packet sojourn time through the queue. This makes the algorithm independent of link rates and reflects the user experience more accurately. Once the minimum queueing delay exceeds a threshold over a fixed time interval, the algorithm goes into the dropping phase. Packet dropping stops only when the queuing delay falls below the threshold. Among its limitations, CoDel requires a timestamp per packet and uses a head drop policy, thus consuming processing and infrastructure resources. Nevertheless, we believe that a buffer sizing algorithm such as WQM may be used in conjunction with an AQM such as CoDel. We intend to explore this interaction in future work.

Algorithm 1: WQM OPERATION PSEUDO CODE

```
1  Set the max. acceptable queuing delay $limit$
2  Calculate the initial $B_{initial}$ based on the current Tx rate ($R$)
   and the round trip delay for a single A-MPDU transmission
   ($ARTT$):
3  $B_{initial} = R * ARTT$
4  for every measurement interval do
5      Calculate queue drain time $T_{drain}$ based on the total
       number of bits in the queue ($BL$) and the percentage of
       time the channel is not busy ($F$)
6      $T_{drain} = \frac{BL/R}{F}$
7      Adjust the queue size $B$ based on whether the network is
       bloated or not
8      if $T_{drain} > limit$ and $B > B_{min}$ then
9          if $alarm_{high}$ is ON then
10             decrease queue size $B$
11         else
12             set $alarm_{high}$ to ON and $alarm_{low}$ is OFF
13     else if $T_{drain} < limit$ and $B < B_{max}$ then
14         if $alarm_{low}$ is ON then
15             increase queue size $B$
16         else
17             set $alarm_{low}$ to ON and $alarm_{high}$ is OFF
```

3. APPROACH

WQM is a practical, adaptive, and lightweight algorithm customized for wireless networks. In this section, we describe the algorithm and show how to choose various WQM parameters.

3.1 Operation

The operation of WQM can be divided into an initial stage and an adjustment stage as shown in Algo. 1. In the initial stage, WQM selects a initial buffer size, $B_{initial}$. This size is calculated using a variation of the Bandwidth Delay Product (BDP) [11]: the buffer size should be greater than or equal to the product of the bottleneck link capacity with effective end-to-end delay. Since it is not always possible to obtain the end-to-end delay, WQM initializes the buffer using a single-hop RTT that is known. Then, it uses an adaptation algorithm to increase the buffer size if the actual RTT is found larger. This should minimize latency for short bursts of traffic and maximize utilization without sac-

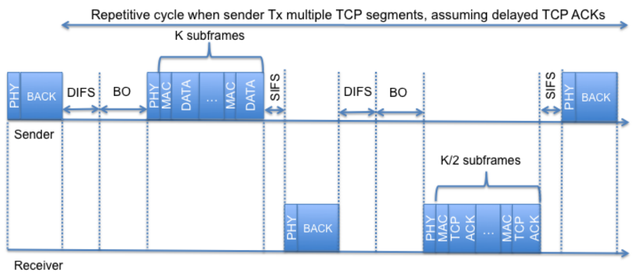


Figure 1: 802.11n MAC overhead per A-MPDU transmission.

rificing latency for longer-running traffic. Hence, the initial queue size can be calculated as:

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

where $B_{initial}$ is the initial buffer size, R is the current transmission rate, and $ARTT$ is round-trip delay for a single A-MPDU transmission as illustrated in Fig. 1.

After assigning $B_{initial}$, the adjustment phase of WQM kicks in. In this phase, the buffer size is tuned to match the current network load. Every given period of time, the queueing delay is calculated using:

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

where T_{drain} is the queue draining time, BL is the queue backlog in bits, and F is the percentage of time the channel is free for the sender to transmit. We divide the queue drain time by the estimate of channel free time to account for the fact that the wireless channel is a shared medium.

If T_{drain} exceeds the predefined maximum *limit* for two consecutive measurement intervals, then this indicates that the buffer is bloated. As a result, the buffer size is decreased to limit the amount of buffering and hence limit the queueing delay. Alternatively, if T_{drain} is lower than *limit* for two consecutive measurement intervals, then the buffer size is increased. Observing the network statistics over two consecutive cycles before taking a corrective action helps account for temporary bursty traffic. This corrective action cannot alter the buffer beyond a minimum and a maximum size, *i.e.*, $B_{min} \leq B \leq B_{max}$, as described in Sec. 3.2 below.

Finally, we note that a larger $B_{initial}$ may be needed to achieve maximum utilization for multi-hop networks. However, we prefer low latency by starting with a smaller than optimal buffer size that will eventually grow in the adjustment phase. As shown in our experimental analysis, this approach works well with long-lived flows. Evaluation with short flows is part of our future work.

3.2 Parameter Analysis

We define upper and lower bounds for the buffer size (B_{max} and B_{min}, respectively) and the maximum allowed queueing delay *limit*. To find B_{max}, consider a 802.11n network with a single TCP stream from the sender to the receiver. Assume that the maximum possible transmission rate is λ packets/s.[1] Assuming that the stream is in the TCP congestion avoidance phase, the TCP congestion window will grow until it reaches W_{max} when a packet loss happens. As a result, the sender halves its TCP congestion window. Hence, it waits for $\frac{W_{max}/2}{\lambda}$ before going to the transmit phase again. The buffer B drain time is B/λ s. The ideal scenario is to have the sender transmitting just before the buffer gets empty to make sure the link is fully utilized, $i.e. \frac{W_{max}/2}{\lambda} \leq B/\lambda$, or

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

Also, to maintain full link utilization, sender transmission rate ($i.e.$`cwnd`$/ARTT$) should be at least λ . Hence, $\frac{W_{max}/2}{ARTT} \geq \lambda$, or,

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

From Eq. (3) and (4),

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

Hence, the maximum buffer size B_{max} is equal to the BDP using the maximum possible transmission rate and the corresponding packet RTT. Mainly, $ARTT$ represents the transmission delay as in wireless networks propagation delay is negligible. Fig. 1 shows the MAC overhead of a single A-MPDU transmission over 802.11n network. As per this figure, $ARTT$ is the sum of the TCP segment transmission time T_{d-DATA} and the TCP ACK transmission time T_{d-ACK} which can be calculated as per the following equations:

$$\begin{aligned} T_{d-DATA} =& T_{BO} + T_{DIFS} + 2 * T_{PHY} + T_{SIFS} + T_{BACK} \\ & + K * (T_{MAC} + T_{DATA}) \end{aligned} \tag{6}$$

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

The system parameters for our 802.11n network are listed in Table 1. A single A-MPDU may contain K TCP segments, for a total size of 64kB or 64 MPDUs, each with its own MAC header. Hence, a transmission duration of $K * (T_{DATA} + T_{MAC})$ is added per A-MPDU. Assuming that TCP delayed acknowledgement is used, only $K/2$ segments are acknowledged. The maximum buffer size is needed when the sender transmits with the highest possible Tx rate (600 Mb/s for IEEE 802.11n) and all frames are sent with maximum A-MPDU length ($K = 64$ subframes). Using Eq. 6 and 7, we calculate the RTT of transmitting a single maximum-sized

[1]We use packets instead of bits for ease of exposition.

Parameter	Value
T_{slot}	slot time = 9μs
T_{SIFS}	Short interframe space = 16 μs
T_{DIFS}	DCF interframe space = 34 μs
T_{PHY}	Preamble and header Tx time = 33 μs
CW_{min}	min. contention window size = 15
CW_{max}	max. contention window size = 1023
T_{BO}	avg. back-off interval
	$= (CW_{min} - 1) * T_{slot}/2$
R	physical rate (Mb/s)
R_{basic}	basic physical rate = 6 Mb/s
K	max. A-MPDU length
T_{MAC}	MAC header Tx time = L_{MAC}/R
L_{MAC}	MAC overhead = 38 B = 304 b
T_{DATA}	data frame Tx time = L_{DATA}/R
L_{DATA}	data frame size = 1500 B = 12000 b
$T_{TCP-ACK}$	TCP ACK Tx time = $L_{TCP-ACK}/R$
$L_{TCP-ACK}$	TCP ACK length = 40 B = 320 b
T_{BACK}	Block ACK Tx time = L_{BACK}/R_{basic}
L_{BACK}	Block ACK frame size = 30 B = 240 b

Table 1: System parameters of IEEE 802.11n [4].

A-MPDU to be about 1.9 *ms* at 600 Mb/s link rate. Using Eq. 5, $B_{max} = 95$ packets. As a lower bound, the minimum buffer size B_{min} should be equal to the A-MPDU length allowed by the link rate. This is because permitting the buffer to be smaller than the number of subframes in a single A-MPDU will result in sending smaller A-MPDUs which impacts the network goodput.

We now compute a lower bound on the allowed queueing delay *limit*. For 802.11n networks with frame aggregation, the maximum allowable A-MPDU size varies with the link rate. Consider a wireless channel with high interference where the rate control algorithm chooses to transmit at the lowest possible data rate (6.5 Mb/s for 802.11n radios). The ath9k release used in our experiments does not support A-MPDU aggregation at 6.5 Mb/s, as transmitting a large A-MPDU at this link rate may violate the 4 ms frame transmit duration regulatory requirement in the 5 GHz band. As a result, *limit* should be greater than or equal to the transmission time of one frame at the lowest possible rate. As per Eq. 6 and 7, *limit* should be greater than or equal to 2.5 *ms*. We tested this value in our testbed experiments over multiple scenarios and found that it allows for significant reduction in latency while preserving overall network throughput.

4. EXPERIMENTAL ANALYSIS

4.1 Implementation Details

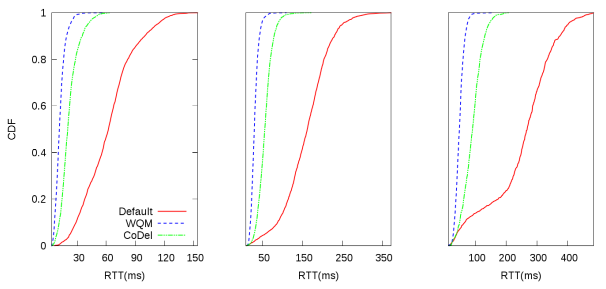
WQM controls the Transmit Queue length *(txqueuelen)* at the queueing discipline (qdisc) using the *ifconfig* utility. An important parameter in WQM is the frequency of obtaining channel statistics. To synchronize WQM with the rate control algorithm, we use the same look-around interval as Minstrel [9], the default Linux rate control algorithm, as the transmit rate stays fixed over this interval. Every 100 *ms* WQM obtains a sample of the current transmission rate, the buffer backlog, and the average A-MPDU length. WQM uses the transmission rate to estimate the queueing time. If this queueing time is longer than the desired target, then queue size is reduced to lower the queueing delay. However, the queue should not be smaller than the maximum number of sub-frames per aggregate as this will only decrease the throughput without delay reduction. On the other hand, if the draining time is less than target, WQM can safely increase the queue size. Our WQM implementation uses a conservative approach in which the buffer size grows linearly (*i.e.*, increases the size by one), but decreases by half. We thus strictly prefer low delay over high goodput. In future work, we plan to evaluate other increase and decrease policies, and their impact on network performance and stability.

4.2 Experimental Setup

We have implemented WQM in Linux and evaluated it in a wireless testbed composed of 10 small form-factor Shuttle computers with Intel E7500 Core 2 Duo processors and 1 GB of RAM. Each Shuttle box is equipped with TP-Link WDN4800 (Atheros AR9380), which is a dual-band, 3-stream MIMO 802.11n wireless card. We use the 5 GHz U-NII (Unlicensed National Information Infrastructure) radio band to avoid interference with our campus production network. The nodes are placed approximately 10 m apart. We patched the stock 3.9.10 Linux kernel with web10g [2] to monitor various TCP statistics of interest, such as congestion window and RTT. The default TCP version in our Linux distribution is TCP Cubic. We run netperf [1] to simulate a large file transfer.

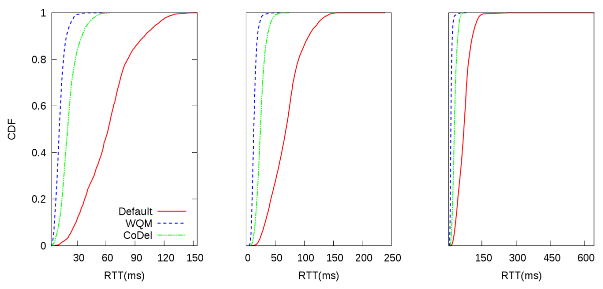
4.3 Experimental Evaluation

We compare the performance of WQM with both the default Linux configuration (*txqueuelen*=1000 packets) and CoDel. We note that WQM and CoDel can complement each other since the former does not implement a selective drop policy. In the first set of experiments, we run a single flow and measure the overall network goodput and latency. To evaluate WQM performance in multi-hop wireless networks, we vary the number of hops between the sender and the receiver from one to three. It is worth mentioning that both WQM and CoDel are deployed at the source and all subsequent relay nodes. The CDF for RTT over various topologies is shown in Fig. 2. The corresponding average goodput is shown in Fig. 3. These results are averaged over three



(a) One hop (b) Two hops (c) Three hops

Figure 2: RTT CDF for a single flow while varying the hop count.



(a) One flow (b) Three flows (c) Five flows

Figure 4: RTT CDF of various concurrent flows over a single hop topology.

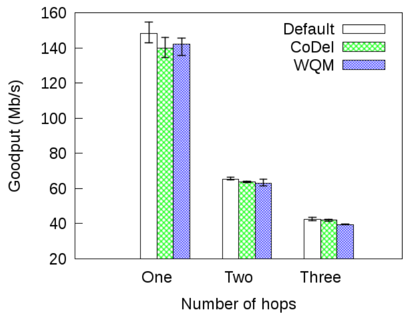


Figure 3: Goodput of a single flow while varying the hop count.

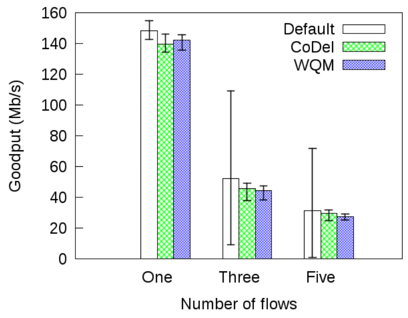


Figure 5: Average goodput with concurrent flows over a single hop topology.

runs and error bars represents max. and min. values. For all scenarios, WQM manages to reduce the network latency by around 5× compared to the 1000 packets buffer and 2× compared to CoDel. For example, in the three-hops scenario, WQM reduces the average delay from 224.4 *ms* using default queues to only 49.47 *ms* at the cost of less than 10% goodput reduction. The three hops average delay with CoDel is 90.428 *ms*, almost twice of WQM with about 5% drop in goodput. We attribute this reduction to the aggressive behaviour of WQM AIMD.

We also evaluate WQM over multi-flow scenarios. We increase the number of concurrent flows from one to three and then five, and measure goodput and latency over a single hop topology. We plot the RTT CDF for the three scenarios in Fig. 4. We also show the average per flow goodput in Fig. 5. Again, bar errors represent max. and min. values over three runs. In all scenarios, WQM reduces the network latency by at least 5× compared to the 1000 packets buffer at the cost of 15% reduction in throughput in the worst case. Compared to CoDel, WQM reduces the delay by almost 2×. Goodput results of WQM and CoDel are within error bounds of each other. Note that as the number of flows increases, the default scheme suffers from severe unfairness (including starvation) between the flows, as reflected by the error bars. Jain's Fairness Index value for the default case is 0.77, compared to 0.99 for both WQM and CoDel. Large buffers in the default scheme lead to severe unfairness because one or more flows can fill up the buffer quickly while starving others. Both WQM and CoDel prevent this behavior by controlling the number of buffered packets.

Finally, we analyze the performance of WQM over both multi-hop and multi-flow scenarios. In this experiment, nodes are organized in a parking lot topology, as shown in Fig 6, where three flows starts simultaneously from the same source but are destined to different nodes in the network. This experiment is repeated several times while enabling and disabling the rate control algorithm, Minstrel. When disabled, the rate is set manually to either 6.5, 13, 65 or 144.4 Mb/s. The average delay per flow as well as the total goodput achieved by the three flows are shown in Fig. 7. Error bars represent max. and min. values over at least three runs. When compared to the default buffering scheme, we find that WQM reduces the end-to-end delay by 8× for the one hop flow, 6× for the two hops flow and at least 4× for the three hops flow regardless of the transmission rate. This reduction in queuing delay does not come at the price of significant goodput reduction. This is another proof that such a large buffer is not needed.WQM

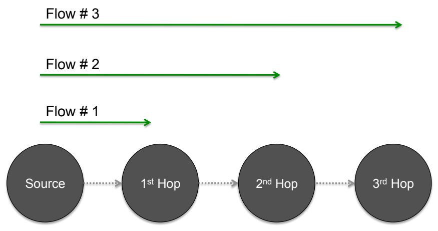
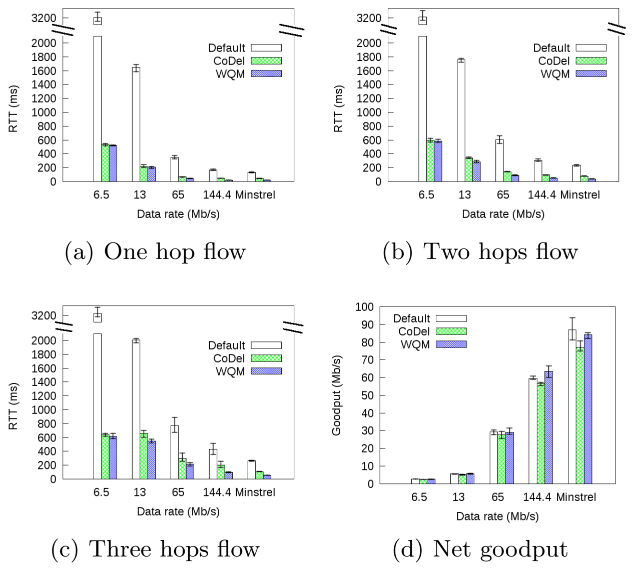


Figure 6: Parking lot topology illustration.



(a) One hop flow (b) Two hops flow

(c) Three hops flow (d) Net goodput

Figure 7: Average end-to-end delay per flow and total goodput in the parking lot topology.

outperforms CoDel in terms of delay and goodput in all the cases. For example, WQM reduces the queueing delay by 2× compared to CoDel when Minstrel is enabled while achieving slightly better goodput.

5. CONCLUSIONS AND FUTURE WORK

Enhancements in 802.11n/ac standards, such as frame aggregation, exacerbate the challenges of optimal buffer sizing in wireless networks. In this paper, we designed a practical, adaptive, and lightweight wireless queue management scheme called WQM. It chooses the queue size based on network load, channel condition, and frame aggregation level. WQM is implemented in Linux and tested on a wireless testbed. Our results show that WQM can reduce the end-to-end latency by a factor of 8× compared to the default Linux configuration. In the worst case, this reduction comes at the cost of less than 15% drop in goodput. Further, WQM outperforms CoDel in terms of delay reduction. Finally, we show that WQM improves flow fairness with respect to the default case.

We are pursuing a number of interesting avenues for future work. We are currently evaluating WQM using a combination of both short and long-lived flows across multiple topologies. We also plan to do a formal analysis of various increase/decrease heuristics in setting the buffer size as used in WQM. Further, we plan to evaluate WQM using flow separation as well as by replacing its drop tail approach with selective drop algorithms. Finally, we would like to study the interaction between TCP pacing and frame aggregation and its consequences for bufferbloat in wireless networks.

6. ACKNOWLEDGMENTS

We thank our shepherd Bob Briscoe and the anonymous reviewers for their helpful suggestions.

7. REFERENCES

[1] Netperf. `http://www.netperf.org/netperf/`.
[2] The Web10g Project. `http://www.web10g.org/`.
[3] J. Gettys and K. Nichols. Bufferbloat: dark buffers in the internet. *Commun. ACM*, 55(1):57–65, Jan. 2012.
[4] IEEE LAN/MAN Standards Committee. *IEEE 802.11 Wireless LAN Medium Access Control and PHYsical layer specifications*. IEEE, 2012.
[5] K. Jamshaid, B. Shihada, A. Showail, and P. Levis. Deflating link buffers in a wireless mesh network. *Ad Hoc Networks*, 16(0):266 – 280, 2014.
[6] T. Li, D. Leith, and D. Malone. Buffer sizing for 802.11-based networks. *IEEE/ACM Transactions on Networking*, 19(1):156 –169, Feb. 2011.
[7] K. Nichols and V. Jacobson. Controlling queue delay. *Queue*, 10(5):20:20–20:34, May 2012.
[8] A. Showail, K. Jamshaid, and B. Shihada. An empirical evaluation of bufferbloat in IEEE 802.11n wireless networks. In *Wireless Communications and Networking Conference (WCNC), 2014 IEEE*, April 2014.
[9] D. Smithies and F. Fietkau. Minstrel rate control algorithm. `http://wireless.kernel.org/en/developers/Documentation/mac80211/RateControl/minstrel`.
[10] D. Taht. What I think is wrong with eBDP in debloat-testing. `https://lists.bufferbloat.net/pipermail/bloat-devel/2011-November/000280.html`.
[11] C. Villamizar and C. Song. High performance TCP in ANSNET. *SIGCOMM Comput. Commun. Rev.*, 24(5):45–60, Oct. 1994.