Law: Towards Consistent Low Latency in 802.11 Home Networks

Yibin Shen and Zili Meng
Hong Kong University of Science and Technology

Abstract

Wireless ultra-low-latency video streaming over 802.11 Wi-Fi networks is increasingly popular, but the latency on the Wi-Fi link is always fluctuating. With the development of CDNs and edge servers, the fluctuation of the wireless last-hop is increasingly dominating the fluctuation of the end-to-end latency. In this paper, we investigate the reasons why the existing Wi-Fi link layer will have a fluctuating latency from a systematic perspective. We find that the hierarchical queueing structure, queue-agnostic rate adaptation, and delay-insensitive retry management of the existing link layer design are the main reasons to a latency spike when channel fluctuates. Thus, we propose **LA**tency-bounded **W**i-Fi (Law), an 802.11 link layer architecture to provide a consistent low latency for the application. Law exploits the loss-tolerance ability from the upper layer video streaming application and significantly avoids the latency spikes caused by the blockage in the link layer at the cost of acceptably additional packet loss. Law maintains a high goodput by carefully redesigning the queueing structure and introducing fine-grained control for each transmission opportunity. We implement the prototype of Law on OpenWiFi and test it with WebRTC – both the tail frame latency and stall rate can be significantly reduced over existing baselines.

1 Introduction

Wireless ultra-low-latency video streaming, such as mixed reality (MR), videoconferencing, and cloud gaming through Wi-Fi networks, are becoming increasingly popular among Internet users due to its convenience. These applications extremely care about the consistency of the low latency. For example, cloud gaming requires an end-to-end latency less than 100 ms, and as reliable as 99.9% or even higher [37, 48, 62]. In the meantime, with the development of Wi-Fi networks, and the deployment of content delivery networks (CDNs) and edge servers, today's network services can provide satisfactory bandwidth as well as *median* latency with Wi-Fi access – the median latency for mobile devices is 25 ms [14].

However, the stringent in consistency conflicts with the nature of bandwidth fluctuation in 802.11 Wi-Fi networks. Since Wi-Fi networks have contention-based connections and unstable channel quality, with the current architecture, they can not provide a consistent low latency all the time – easily, the latency just in the Wi-Fi last-hop will rise to even more than 100 ms [19, 67, 72, 82]. This issue is outstanding in the whole end-to-end path as previous efforts show that Wi-Fi last-hop transmission accounting for more than 90%

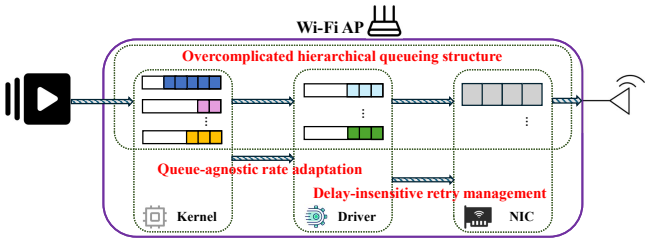


Figure 1: Wi-Fi link layer design is not systematically suitable for low-latency applications.

of the end-to-end latency fluctuation (§2.1, [27, 60]). Moreover, this can be increasingly felt as the median RTT continues to decrease and the latency fluctuations in the rest of the transmission system stabilize. Previous reports showed that users typically could reach the nearby datacenters or edge servers within 10 ms (one-way transmission latency, which is approximately half-RTT) [13, 22]. Especially for ultra-low latency video, such as cloud gaming, one of the latest papers indicates that in 90% of cases, the base RTT is less than 20 ms [73]. In other words, wireless networks are increasingly dominating the entire end-to-end transmission loop.

The key reason is that the Wi-Fi link layer is not systematically designed for low latency in different perspectives:

- **Overcomplicated hierarchical queueing structure.**: Designed for maximizing the throughput, 802.11 systems have a hierarchical queueing structure similar to the multi-level caching. The MAC subsystem, Wi-Fi driver, and Network Interface Card (NIC) hardware all have their own queues and independent schedulers. With the stringent latency requirement, controlling multiple queues with packets accumulated at different places is difficult.
- **Delay-insensitive retry management.** When the packet transmission is corrupted, the 802.11 driver will retry to send it several times before it gives up and lets the upper layer handle the packet loss. However, the existing retry management is not aware of the delay for the packet. For example, Qualcomm's ath9k driver will keep retransmitting the same packet for 31 times before giving up the transmission [6, 9], which can take up to hundreds of milliseconds. What's worse, this will further block the transmission of the subsequent packets, leading to stalls.
- **Queue-agnostic rate adaptation.** 802.11 needs to adapt the data rate based on the channel conditions [20, 76]. For example, when the channel quality degrades, the sender will decrease the modulation coding scheme (MCS) to increase the success rate at a cost of a lower data rate. However, determining the data rate only based on network con-

dition is insufficient – blindly reducing the data rate will result in bufferbloat in the queue before transmission, incurring a high end-to-end delay.

There are a number of research efforts reducing the latency for the Wi-Fi link layer, such as flow scheduling including Wi-Fi Multimedia (WMM) [1, 11], and some research work [43, 57, 65]. However, centralized at these designs is to *prioritize* low-latency flows over other flows rather than providing a consistent low latency. For example, when the channel sharply fluctuates and the flows with the highest priority still can not send through, all the issues mentioned above still can not be mitigated. There are also some individual efforts that optimize latency by rate adaptation [55], transmission opportunity (TXOP) [15], and active queue management (AQM) [17, 32, 33, 64, 66]. But they are mainly algorithmic level optimizations and still suffer from the issues above. These methods still introduce hundreds of milliseconds of latency at the tail, even when there are no competing flows (§4, [26, 40]).

Our key observation is that for ultra-low-latency applications, getting dropped at the link layer might be more beneficial than getting blocked. Ultra low-latency video streaming is, in fact, loss-tolerant at the application layer to a certain extent. Error concealment mechanisms, including forward error correction (FEC) [61, 69, 75, 79] and loss-tolerant video codec [25, 49, 52, 74], provide great recovery ability regarding packet loss. The reduction of RTT and latency-sensitive congestion control algorithm (CCA) [16, 21, 44, 68] also makes the cost of retransmission at the link layer and transport layer comparable. By this way, we could reduce tail latency significantly with little cost, resulting in overall benefits at the application layer.

Therefore, we propose Law, which implements a **LA**tency-bounded **W**i-Fi link layer design for low-latency video streaming. Law can ensure that each packet stays at the link layer for no longer than a time limit by discarding packets that are likely to result in blockage and high end-to-end latency. Law enforces the principle of bounded latency to the queueing structure, retransmission control, and rate adaptation of the 802.11 system.

However, it is non-trivial to maintain the goodput in the same time of providing a bounded latency. Simply dropping too many packets will lead to massive retransmissions and not result in benefits in end-to-end latency. Therefore, we have to carefully maximize the goodput. We first break down the hierarchical queueing structure and merge all the queues into one so that we can easily manage the packets that are severely delayed. With this unified queue, we jointly adjust the data rate and retransmission limit based on channel quality, queue length, and packet latency variations. We also precisely control at a finer granularity of per transmission opportunity rather than periodically adjusting the parameters. We further adjust the packet loss rate and loss pattern to maximally utilize the loss-tolerance ability from the upper layer. Law can achieve an optimal effect for real-time services with low RTT and a certain degree of loss-tolerance, including but not limited to applications from CDNs and edge servers.

We deploy the proposed latency-bounded link layer on OpenWiFi [4, 47] (around 2K lines of C and Verilog codes) and conduct real-world experiments with WebRTC [12]. And results in §4 show that Law can reduce 99.9ile per-packet latency by 75.3% to 83.2% compared to baselines while controlling the packet loss to below 1% accurately. From the application layer, for low-latency video streaming, Law also reduces the 99ile frame delivery latency by 50.6% to 70.3%. Even when compared to much more powerful commercial routers, Law can still reduce the 99.9ile tail latency by 46.0%. In addition, we demonstrate Law's robustness in handling different scenarios such as TCP BBR, traffic with competing flows, and fairness.

Our main contribution can be summarized as follows:

- We illustrate that the overcomplicated hierarchical queueing structure, queue-agnostic rate adaptation, and delay-insensitive retry management in the existing Wi-Fi system are the root cause of latency fluctuation.
- We simplify the queueing structure in the Wi-Fi link layer, retaining the management only in the Wi-Fi driver as well as minimizing the hardware queues to make latency control efficient and reliable. We co-design a fine-grained latency-sensitive data rate and retry adaptation algorithm.
- We implement Law over the OpenWiFi hardware and evaluate with WebRTC. Extensive real-world experiments show that Law can effectively control the packet-level latency and improve application-level performance.

2 Background and Motivations

2.1 Tail Latency in Wi-Fi

Ultra-low-latency video streaming requires delivering video contents with minimal delay, typically under 100 milliseconds [37, 48]. To achieve that, various research works on latency optimization have been proposed in the application layer, including packet loss recovery, latency-sensitive CCAs, frame pacing [7], etc. And ultra-low-latency streaming is mostly based on edge server (e.g., cloud gaming), peer-to-peer connection (e.g., WebRTC, screen casting), or multipath transport system [29, 81, 82], which further promotes applications to meet ultra-low-latency requirements.

Even as upper layers have been largely enhanced, the latency performance with Wi-Fi access is still unacceptable. We collect some previous measurements in Table 1 – various works have shown that it is not uncommon for a Wi-Fi network to fluctuate up to 100ms latency for a single packet. As long as the base RTT is as low as 10-20 ms, Wi-Fi last-hop transmission is dominating the overall end-to-end latency.

To demonstrate this, we also conducted fortnight-long experiments in our campus from an edge server with the same experimental setting in §4.1 to see exactly how much the

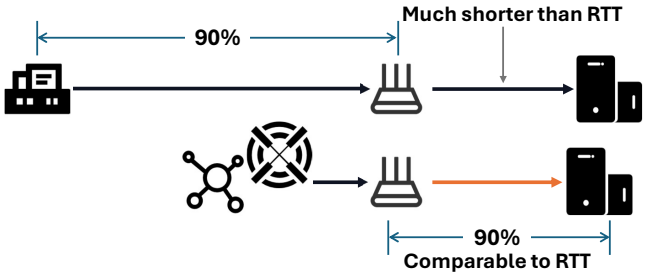


Figure 2: Wi-Fi last-hop transmission is increasingly being the bottleneck in ultra-low-latency streaming.

Packet-level performance	
ABC [40]	Extreme per-packet delays over 300ms are observed at 95ile.
QAir [67]	Wi-Fi last-hop RTT can be higher than 100ms at 80ile.
Augur [82]	99.9ile Wi-Fi RTT goes above 150ms even in the 5G band.
Frame-level performance	
Hairpin [61]	0.2% of frames suffer high latency more than 100ms.
Augur [82]	Frame delivery latency at the 99.9ile exceeds 200ms.

Table 1: Poor latency performance of Wi-Fi last-hop

wireless last-hop transmission contributes to the overall latency. The access point (AP) used here is a TP-LINK TL-WDR4310 (refreshed with OpenWRT 23.05.0) [10]. We measure the proportion of the Wi-Fi last-hop transmission over the end-to-end latency and present the result categorized by different end-to-end latency in Fig. 3. Wi-Fi last-hop becomes the primary contributor to the latency when the end-to-end latency increases. And when the overall latency gets higher than 70 ms, the Wi-Fi last-hop generates more than 95% of the overall latency in most cases. Just as illustrated in Fig. 2, latency in Wi-Fi networks has become more and more comparable to the end-to-end latency. And Wi-Fi last-hop transmission are increasingly being the bottleneck in the end-to-end transmission of ultra-low-latency streaming.

Wi-Fi tail-latency breakdown. We further conduct a latency breakdown of the Wi-Fi last-hop latency in Fig. 4. As the latency increases, queueing delay, including three queues in Fig. 1, and transmission time (airtime plus management intervals) both increase a lot. For latencies above 70 ms, queueing delay contributes for more than 80% on average. And the other part includes latency from the Linux kernel and IP stack, etc. Latency produced in the link layer becomes more and more critical when the overall number gets higher. In this case, the effect of existing solutions for the higher layers [40, 60] is limited.

2.2 Design Gaps in Wi-Fi Link Layer

Our insight is that the Wi-Fi link layer is systematically not suitable for ultra-low-latency streaming.

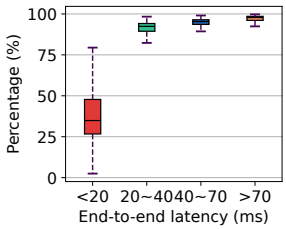


Figure 3: The proportion of Wi-Fi last-hop transmission in end-to-end latency. The median proportion of the last-hop reaches above 95% when latency increases.

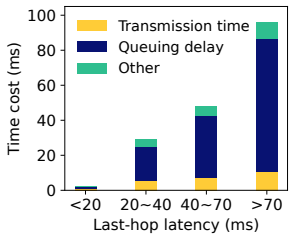


Figure 4: Wi-Fi last-hop latency breakdown. Queueing delay and transmission time are the main contributors to high tail latency.

Overcomplicated hierarchical queueing structure: We introduce the queueing structure in OpenWRT and Qualcomm's ath9k driver as the state-of-the-art open-source system for Wi-Fi routers, as shown in Fig. 5.

First, packets are enqueued into the flow queues maintained by the MAC80211 subsystem in the form of a MAC Protocol Data Unit (MPDU) and wait for the lower-layer driver to pull. When needed, the driver will actively pull packets from the flow queues for further transmission. Meanwhile, the link layer data transmission rates are optimized here for each packet. The driver keeps retry queues to store packets that have failed in previous transmissions in hardware, a.k.a. software retransmission. The driver then performs frame aggregation and header packing to form a data frame (a.k.a,, Physical Layer Service Data Unit, PSDU). The PSDUs are then written to the hardware queues. The maximum length of a hardware queue is 8.

Such overcomplicated hierarchical queues incur latency for different reasons, e.g., arrival rate and flow competing in flow queues, channel contention in hardware queues, etc. And we list the specific reasons for each layer in Fig. 6. Except for the statistics from Fig. 4, we further illustrate that queueing delay is the main contributor to the high tail latency using real-world traces. As Fig. 7 illustrates, the queueing delay increases dramatically to 80 ms when channel status fluctuates, but the reduction of the queue delay is quite slow. In conclusion, hierarchical queues have little awareness of each other and is challenging to be coordinated to react to rapid changes.

Delay-insensitive retry management: We also picture the retransmission process in Fig. 6. Retransmissions in the Wi-Fi link layer have two types: software retransmission in the driver and hardware retransmission in the hardware.

- `Hardware retransmission:` If the NIC hardware fails to receive the ack for the current PSDU after a timeout (typically less than 1 ms), the hardware will start backoff contention and retry for a certain times. The limit of the number of hardware retransmissions is set based on the

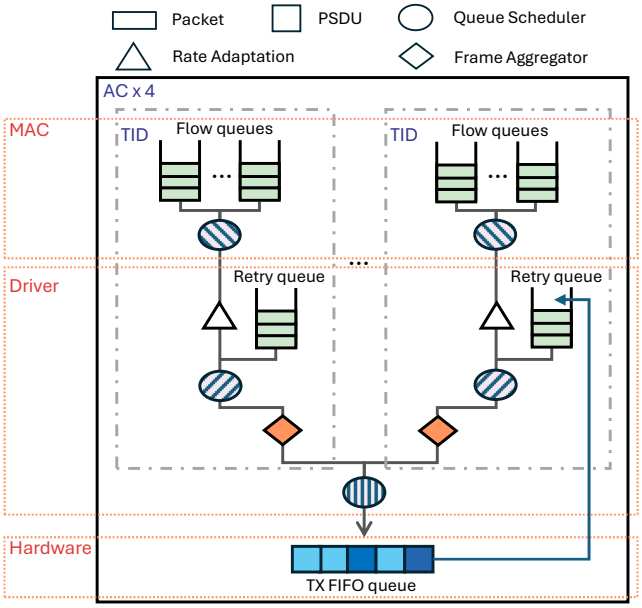


Figure 5: A brief introduction of the queueing structure in `ath9k`.

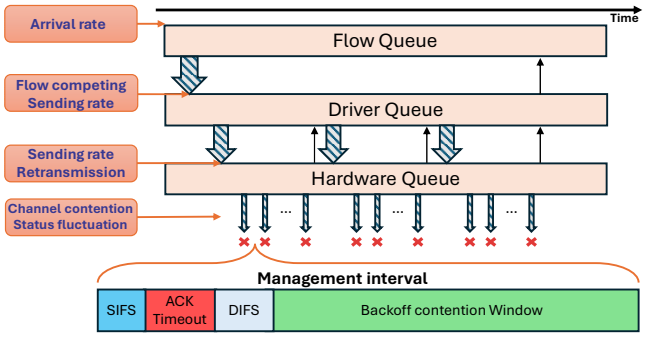


Figure 6: Transmission scheduling in `ath9k` and reasons for high latency in each layer.

data rate. When the number of hardware retransmissions reaches the limit, the packet is returned to the software retry queue in the driver.

- `Software retransmission:` When the failed PSDU is returned to the software retry queue, there will be additional rounds of software retries. Packets are dropped only if the total number of hardware retransmissions is greater than a threshold, which is 31 times in ath9k. So packets will have a long delay before getting discarded.
- `Out-of-order delivery:` The hierarchical packet scheduling also brings out-of-order delivery. If the earlier frames fail after several hardware retransmissions while the subsequent frames succeed, packet loss has already been detected on the client due to the out-of-order delivery.

Moreover, simply setting a lower number threshold for the number of retransmissions does not work. As shown in Fig. 8, we tested three signal strength (RSSI) that are common

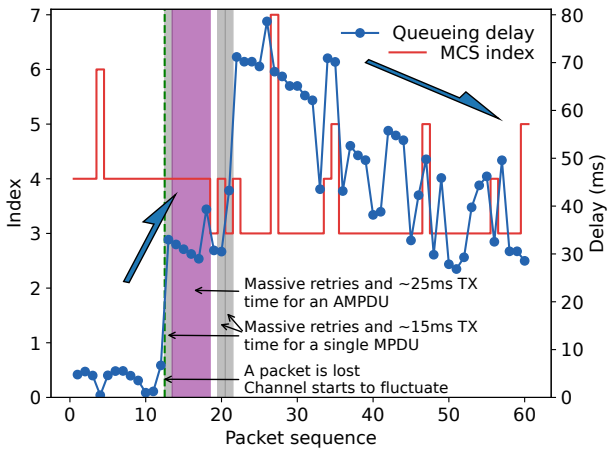


Figure 7: A trace that shows hierarchical queues and queue-agnostic RAA can not cope with rapidly increasing latency.

in daily life [63, 72]. The percentage of packets retransmitted varies a lot at different RSSI by up to 15×. Furthermore, we present another case in Fig. 9 to show how the delay-insensitive retry management harms transmission latency. Several PSDUs take more than 10 retransmissions and consume dozens of milliseconds for transmission time. The impact of packet loss over video stalls was also unveiled by previous production-level measurements [82].

Queue-agnostic rate adaptation: As a higher data rate tends to lead to a higher probability of packet loss in physical transmission, rate adaptation algorithms (RAAs) aim at choosing an appropriate rate under the current channel status. Most work on rate adaptation focuses on maximizing goodput [8, 24, 42, 51], while a few of them pose optimization regarding energy-efficiency [54], extremely low loss rate for gaming devices [78].

However, the link layer RAAs proposed for Wi-Fi APs have two problems in providing consistent low latency:

- They are coarse-grained. Existing RAAs only set the data rate once for each packet or periodically and rely on retransmissions to make up for inaccurate rates, which is insufficient for rapid fluctuations at the transmission opportunity level.
- They are network-aware but queue-agnostic. Existing RAAs mainly focused on channel capacity to adjust the sending rate. But this will backpressure the packets to the queue backlog on the 802.11, leading to high latency. And looking back to Fig. 7, the RAA has been more conservative after the 13th packet, but also contributed to the spike of the queueing delay.

2.3 Loss-tolerance in the Upper Layers

Our key observation is that for ultra-low-latency applications, getting dropped at the link layer can be more beneficial than getting blocked. The status-quo frameworks have provided packet recovery abilities and improved loss-tolerance

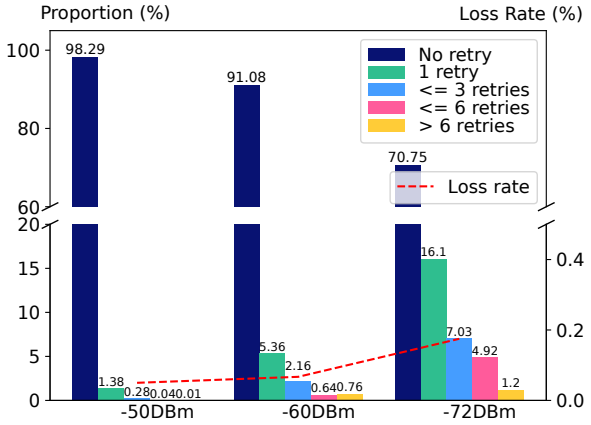


Figure 8: The relationship between the number of retransmissions, packet loss rate, and RSSI. Wi-Fi performs a lot of retransmissions to ensure a relatively low loss rate, which results in high latency.

at the application layer and transport layer [34, 45].

Forward error correction: FEC is a widely used technique to recover losses. Existing FECs used in ultra-low-latency streaming are implemented by means of redundant coding, reproducing lost packets without retransmission. Webrtc (the state-of-the-art open source low-latency streaming platform) provides three coding methods in rows, columns, and 2D arrays [79]. We use WebRTC to test the performance of FEC with no additional packet loss and 1% additional packet loss. The test settings are described in §4.1, and the FEC redundancy ratio is fixed to one eighth, which is a commonly used rate in previous work [53, 61]. We present the results in Fig. 10. FEC could easily cope with the additional loss with a mere increase of no more than one RTT in 99.9ile latency.

Error concealment in video codecs: There have been a number of efforts in both industry and academia to improve the packet loss resistance of codecs. The error concealment is a common technique used in the state-of-the-art codecs, e.g., H.264 [49, 52, 74]. It requires the encoder to reduce the compression rate and provide some information redundancy to allow the encoder to reconstruct the lost data with certain algorithms to ensure frame integrity [56]. Nowadays, there is also some work on frame skipping [36] or neural codecs [25, 77] specially for loss-tolerant ultra-low-latency streaming.

Interaction with congestion control: Moreover, slight packet losses will not lead to bitrate decrease of CCAs in most of ultra-low-latency video streaming. Ultra-low-latency streaming usually uses latency-sensitive CCAs that aim to keep buffers in networks clean and utilize latency increase as the signal to reduce the sending rate. Meanwhile, latency-sensitive CCAs are highly loss-tolerant. For example, GCC can tolerate without rate decrease 2% packet loss [28], while BBR can tolerate with at least 5% random packet loss [21].

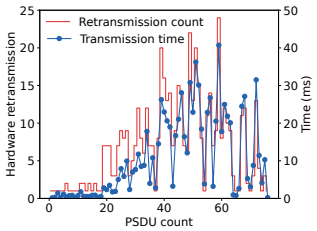


Figure 9: A trace that shows massive hardware retransmissions cost a lot of time.

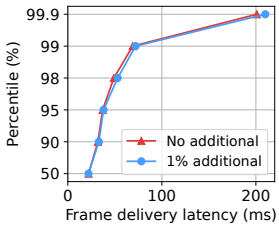


Figure 10: Frame delivery latency with different loss rate. 1% loss does not affect much due to FEC.

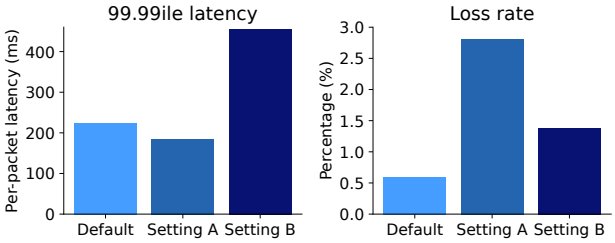


Figure 11: Naive modifications lead to negative optimization. Setting A and B are introduced in §3.1.

3 Law Design

In this section, we introduce our Law Wi-Fi link layer design.

3.1 Basic Idea and Challenges

Considering existing Wi-Fi link layer designs and state-of-the-art ultra-low-latency streaming systems, we proposed our basic idea of making use of the loss-tolerance in the upper layer to reduce the latency in the Wi-Fi last-hop. We aim to implement a latency-bounded link layer by dropping packets that are perceived to be or have already suffered high latency. Simply send fast when latency dramatically grows instead of trying to be reliable and make sure all the packets stay in the link layer for no longer than the latency bound. We allow reasonable packet loss and wait for the application layer to recover. However, to realize this, there are still a few challenges ahead.

Queueing latency is hard to control: As we've shown in Fig. 7 and Fig. 9, even a single packet may lead to dozens of milliseconds of latency. This kind of latency affects all three layers of queues and brings the following technical problems:

- Controlling queue length will not work for Wi-Fi networks. In situations where a single packet can cause latency to fluctuate badly, queue length is no longer an efficient controller of latency.
- Hierarchical queues lead to a discrete distribution of latency, and managing latency for each queue separately can be harmful. Applying a latency bound for a particu-

lar layer of queues can not address latency jitters and can even significantly affect the overall performance.

We made a simple modification to OpenWiFi to test how naively applying latency limits to the current queueing structure will affect the loss rate and throughput. We measure the performance on packet-level and the testbed used is the same as in §4.1. We apply a limit of 25ms to flow queues in the MAC layer and limit the sum of the length of the four AC queues to 8 packets in hardware in Setting A. We present the results in Fig. 11. Even with a terrible 8.0× increase in loss rate compared to the default setting, the 99.99ile per-packet latency of Setting A is still nearly 200ms.

Aggressive structure revision may lead to negative optimization: To address the issues caused by the hierarchical queueing structure, it is easy to think of simplifying the current structure. But it is not as simple as it seems. In Fig. 11, we simply disable hardware queues for Setting B. And we can see a more than 400 ms 99.99ile per-packet latency and a 2.5× increase in loss rate compared to the default setting. The main reason behind this is that we sometimes have to wait for an interrupt and communication between the hardware and software to pull packets from the driver before transmission if we remove the hardware queue. During this period, it is highly likely that the initially idle channel is occupied by other devices. That would lead to massive missed transmission opportunities and performance degradation in the contention-based Wi-Fi channel.

Active packet dropping needs to be carefully controlled: We expect to counter the fluctuations of the wireless channel by sending packets fast or even dropping packets to ensure a consistent low latency. But still, there are two key points in the system design that must be concerned with or we will not be able to achieve the improvement we desire.

- Make sure that dropped packets are within the upper layer's recovery capabilities. Loss recovery techniques are typically executed within a frame or a group of packets. For example, interleaved XOR FECs [79] are encoded by rows and columns in a group of packets, and the maximum number of consecutive packet losses that can be recovered is the number of columns. Reed-Solomon FECs [75] divide data packets and redundancy into blocks of the same size, then decode the desired lost data blocks. However, the number of recoverable lost packets in a single video frame is still limited by the redundancy ratio. Extra end-to-end retransmissions are definitely not what we want. So, not only is the loss rate what we have to control, but the loss pattern also matters a lot.
- Precisely drop packets as soon as the queue starts to grow. Our ultimate goal is to avoid high latency, not to start reacting after it occurs. So we not only focus on network capacity, but also the queueing latency variations and the length of building-up queues. A latency-sensitive, rapid, and precise packet scheduling mechanism needs to be proposed.

3.2 Framework Overview

The key objective of Law's design is to provide a bounded latency while still trying to maximize the goodput with fine-grained control. The fine granularity has two perspectives:

- Instead of modularized queue control for each layer individually, we integrate all the queues into one big queue and carefully control the overall queueing delay for each packet (§3.3).
- Instead of coarse-grained parameter controls such as fixed retransmission limits carried by each data rate, we update the sending rates and retransmission limit for each transmission opportunity (§3.4).

The overview of Law's link layer structure can be simplified as Fig. 12. The rest of the section describes the two parts of our design in detail.

3.3 Queueing Structure Reshaping

Our solution is to merge flow queues, retry queues, and a part of the hardware queues together. We can also tell from Fig. 12 that we moved the flow queues down to the driver and no longer keep the retry queues in the driver. In hardware, we only store at most two PSDUs across four ACs at the same time. In this way, we partly merge hardware queues into driver queues.

For packets that are returned from hardware to driver for software retransmission, we push them immediately back to the hardware queue without queueing (during which our joint rate and retransmission control will be made, we leave the details in §3.4). By default, every packet could only have one time software retransmission. Such retransmission scheduling, plus keeping at most two PSDUs in the hardware, also minimizes the negative effect of out-of-order delivery.

With such a reshaped queueing structure, we can manage latency quite wisely. We set a threshold $Limit_q$ for the latency of packets in the driver. Any packets with more than $Limit_q$ latency in the driver will be dropped. In the current hierarchical queues, we have to divide the time limit into three layers, e.g., 10 ms each. Then we can achieve an overall 30 ms latency bound. However, there are many cases where a packet is dropped even though its overall latency does not exceed 30 ms (Fig. 13). However, all three cases in the figure can be transmitted if we only have one queue with a 30 ms threshold. Centralizing latency and controlling latency by one queue can minimize the unnecessary loss.

We give a simple explanation about the additional loss brought by hierarchical queues in Fig. 14. L_{single} denotes the latency in a single layer's queue, L_{total} denotes the cumulative latency across three layers. We can think of the latencies generated by each of the three queues as independent variables. And their sum will result in a more concentrated distribution of latency values, which is the distribution of overall latency. Thus, the number of packets with latency

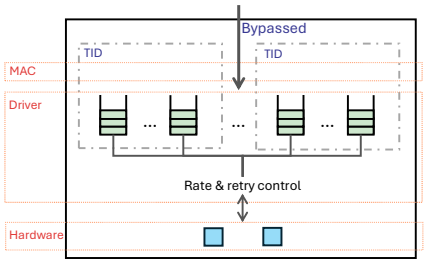


Figure 12: Overview of Law.

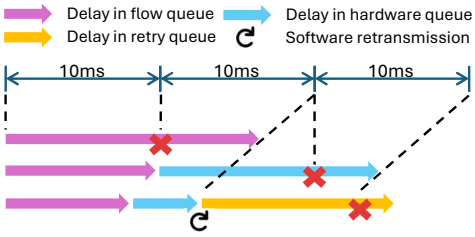


Figure 13: Latency bounds in hierarchical queues lead to unnecessary loss.

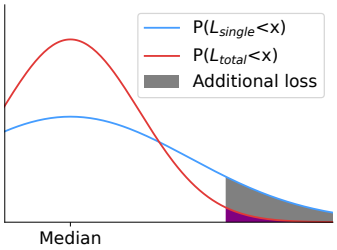


Figure 14: Integrated queues provide a more centralized latency distribution.

higher than the threshold is reduced.

Maximize transmission efficiency: However, it is non-trivial to merge all the queues together. As introduced above, Law still stores two PSDUs in hardware. One important reason to maintain some packets in the hardware is to maximize every transmission opportunity as long as it is available in the channel (§3.1).

Moreover, it is not wise to simply use the same latency bound across driver and hardware. Let us say that we use simply an overall threshold $Limit_q$. One packet may enter the hardware and start transmission after $Limit_q/2$. But the transmission keeps failing due to channel fluctuation, and the packet will still occupy the rest $Limit_q/2$ for hardware retransmissions, resulting in blockage and increased queueing delay for the following packets. It is definitely much better to end transmission early and return the packet back to the driver to re-select a data rate for this packet.

So our response to this is to maintain two separate thresholds in driver and hardware. We set a time limit $Limit_h$ at the packet level (PSDU) in hardware. The TX time (counting from the time when the PSDU starts to be sent by the hardware, including management intervals, backoff contention windows, and airtime) of each PSDU must not exceed this limit. Once the limit is reached, we stop hardware transmission and return the packet back to the driver. In this way, the latency of all packets produced by the Wi-Fi link layer can be kept below $Limit_q + 2 * Limit_h$. Since the length of driver queues is often much larger than 2, $Limit_q$ will be much larger than $Limit_h$ too. In this way, Law maximizes transmission efficiency while still ensuring centralized latency in the driver.

3.4 Rate and Retransmission Control

We try to let packets pass quickly when the queueing delay starts to increase or the number of waiting packets exceeds the network capacity, instead of keeping retransmitting them. When channel status fluctuates, a too low data rate will lead to queue accumulation. On the other hand, a too high data rate may result in massive retransmissions, which also lead to queue accumulation. However, with consideration for retransmission limit adaptation, e.g., by choosing a relatively high rate and a strict retransmission limit at the same time, this problem can be solved perfectly with reasonable packet loss. Therefore, we strive to balance and select the optimal rate and retransmission limit.

We first control the packet loss rate and maximize throughput by a fine-grained RAA that re-selects the rate for each software retransmission. Secondly, to keep the queue latency at a low level, we further co-design a latency-sensitive rate and retransmission control strategy for each transmission opportunity.

Global stats	Explanation
$Limit_q$	latency limit in driver queue
$Limit_h$	latency limit in hardware
Len_{max}	Limit of the number of packets stayed in driver
$State$	State of rate & retransmission control
$Rate$	Current data rate
$Rate_{prev}$	Previous data rate
$Rate_{min}$	Data rate with the minimum average TX time
T_n	EWMA of TX time for all transmitted packets
T_i	EWMA of TX time for packets transmitted by rate i
S_n	EWMA of transmission success rate for all transmitted packets
S_i	EWMA of transmission success rate for packets transmitted by rate i
N_{suc}	Number of consecutive successful transmissions
N_{ret}	Number of consecutive software retransmissions
N_{prob}	Number of failed probing
Thr_p	Threshold for start probing

Table 2: Parameters used in rate and retransmission control

Parameters calculation: We summarize the parameters we need in Table 2. Here, we introduce how we calculate the parameters. First of all, we keep the exponential moving average (EWMA) of TX time T_n and software transmission success rate S_n for all packets during runtime. Calculating software transmission success rate instead of hardware transmission means we focus on whether the current rate could send the packet to the receiver within $Limit_h$. We also calculate the EWMA of TX time T_i and software transmission success rate S_i for each data rate with index i.

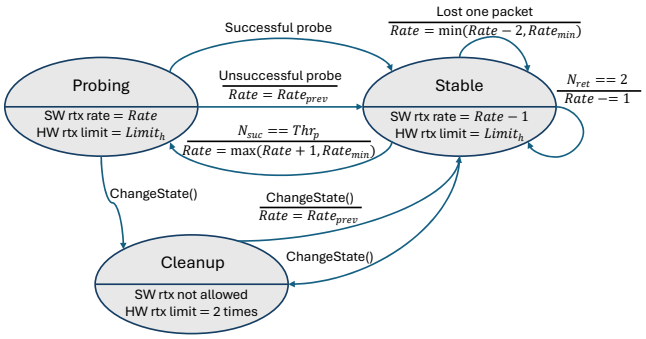


Figure 15: Overview of rate and retransmission control in Law.

Based on T_n and S_n, we calculate Len_{max} as shown:

$$Len_{max} = \frac{Limit_q * S_n}{T_n} \quad (1)$$

Len_{max} is used as a threshold to indicate that we should start reducing queue length by the latency-sensitive rate and retransmission control.

Besides, whenever we successfully decide on a new rate, we reset the number of consecutive successful transmissions N_{suc}, the number of consecutive software retransmissions N_{ret}, and the number of failed probing N_{prob} to 0.

They will be updated every time a packet is returned from hardware. And if a software transmission fails, N_{suc} will be set to 0. In contrast, if a first-time software transmission succeeds, N_{ret} will be set to 0. Finally, Thr_p denotes the threshold to start a new probe, which means if $N_{suc} >= Thr_p$, we will be able to probe a higher data rate. It is calculated by:

$$Thr_p = (N_{prob} + 1) * 5 * IndexOfCurrentDataRate \quad (2)$$

Latency-sensitive rate and retransmission control: Our data rate and retransmission joint control algorithm could be divided into three states: stable state, probing state, and cleanup state. Stable state is the initial state, and is also the state where we execute normal data rate adaptation, rate probing, and retransmission strategies.

When the channel is stable, we enter the probing state and try to probe higher data rates to increase throughput. We use two packets for probing and judge the success of the probing by the following two conditions.

- No packets are lost, and at least one packet is successfully transmitted without software retransmission.
- The average TX time for these two packets is lower than the T_i of $Rate_{prev}$

The cleanup state is the most important state where we actually deal with growing queue latency and poor channel capacity. We enter the cleanup state when either of the following two conditions is met:

- A packet in the driver or just returned from the hardware has a latency higher than $Limit_q$ in the link layer.
- The number of packets currently stored in driver queues exceeds Len_{max}.

The detailed decision-making function is shown by Algorithm 1. In the cleanup state, we allow two hardware transmissions for each software transmission, and no software retransmission. And if two consecutive packets are dropped, we turn down the $Rate$ by one index. On the other hand, Thr_p is fixed to $5 * IndexOfCurrentDataRate$. If the number of consecutive successful transmissions equals Thr_p, we increase the $Rate$ by one index. The overall control flow of Law's latency-sensitive rate and retransmission control is illustrated by Fig. 15.

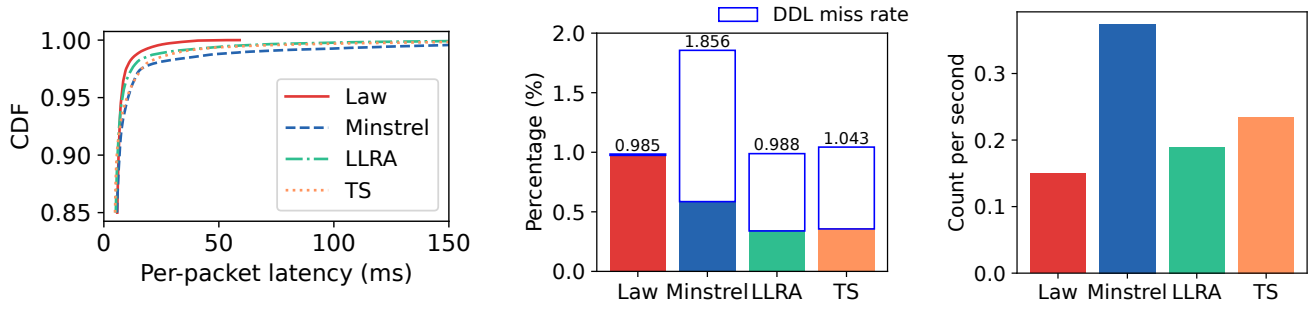
Handling loss pattern: Avoiding consecutive packet loss is crucial in our design. To ensure that packet loss is distributed within the resilience of the upper layer, if there are three consecutively lost packets, we add one more software retransmission for the current packet as a double insurance. Besides, packet loss is most likely to occur in the Cleanup state, as retransmission limits are more stringent. So we use a more sensitive adjustment to improve the loss control ability in the Cleanup state as described above.

Algorithm 1: ChangeState() Function

```
// Called when a packet enters the driver
   or is going to be pushed into hardware
Input: Transmission status of the packet
if packet comes from hardware ||
packet will be pushed into hardware then
    update Len_max, T_n, T_i, S_n, S_i
    if packet's latency> Limit_q then
        drop packet
        Rate_prev = Rate
        State = Cleanup
    else if State = Cleanup &&
    current packets in driver < Len_max then
        Rate = Rate_prev
        State = Stable
else
    if current packets in driver> Len_max then
        Rate_prev = Rate
        State = Cleanup
```

4 Evaluation

We implement Law in OpenWiFi on SDRPi with XC7Z020 chipset [3] and AD9631 radio-frequency transceiver [2]. OpenWiFi is a Linux Mac80211 compatible full-stack Wi-Fi design based on software defined radio (SDR) that currently supports 20 Mhz, 802.11N data rate, and single-in-single-out (SISO) for 5G Wi-Fi connection. We chose OpenWiFi because it is an open-source Wi-Fi platform that allows direct control over hardware. Law requires direct control over hardware to achieve the dynamic hardware retransmission control. We skip the original three-layer queues in the MAC layer, driver, and hardware, replacing them with the queueing structure proposed in Law. The rest of the control flow in the link layer (e.g., header encapsulation) stays unchanged. Law overrides the rate and retransmission decisions before



(a) CDF of per-packet latency. (b) Average loss rate and deadline miss rate. (c) Frequency of consecutive losses.

Figure 16: The performance of Law and three baselines on packet-level. Law significantly reduces tail latency. And Law has the lowest sum of loss rate and deadline miss rate, and the fewest consecutive packet losses.

packets enter the hardware. Regarding possible future deployment for device vendors, we leave the discussion in Appendix A.

In this section, we evaluate Law by real-world experiments in the following aspects:

- **Packet-level.** We ensure that Law optimizes packet-level latency and controls packet loss well to show that Law is theoretically applicable to low-latency applications. We verify that Law actually provides bounded latency in the link layer and test its abilities in controlling packet loss. Law could achieve up to 83.2% latency improvement at 99.9ile while producing the fewest consecutive losses.
- **Application-level.** We then measure the improvement of Law on end-to-end frame delivery latency for low-latency video applications. We compare Law with some RAAs proposed in previous research works, as well as some commercial routers. Law could reduce 99.9ile frame delivery latency by more than 58.4% compared to OpenWiFi baselines and 18.6% to 46.0% compared to commercial routers.
- **Microbenchmarking.** Besides, we conduct some microbenchmarkings to see how Law performs when varying the experimental targets, including changing parameters and adding a competing flow. We also measure Law's performance of TCP BBR flows to demonstrate Law's robustness under different traffic types. Finally, we evaluate Law's impact on fairness regarding different CCAs.

4.1 Experimental Setup

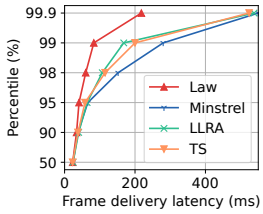
Testbed. For packet-level experiments, we use Iperf to send UDP flows with the bitrate fixed to 10 Mbps. The Iperf sender is run on an OpenWRT router, which is connected to the OpenWiFi AP by Ethernet directly. As for application-level experiments, we use WebRTC (M119) as the sender and receiver of low-latency video streaming. The target bitrate and framerate are set to 10 Mbps and 30fps. And the WebRTC sender is deployed at an edge server for realistic end-to-end transmission. Both Iperf and the RTC receiver are run on a MacBook Pro with MacOS 15.3.1, which is connected to the testing OpenWiFi AP within the 5G channel. And the $Limit_q$ and $Limit_h$ are set to 40ms and 3ms as a default setting. We conducted experiments during office hours in our laboratory, typically from 1:00 p.m. to 7:00 p.m. The position of the tested AP and receiver remains unchanged during the whole evaluation. Due to a certain degree of personnel turnover within the office, the results obtained under such an experimental setup can represent the overall performance under different levels of channel interference. We ran each test for at least ten minutes and cycled through the baselines to ensure that the result is fair.

Baselines. We used OpenWiFi with the three layers of queues mentioned above and implemented some related and frequently noted RAAs as baselines.

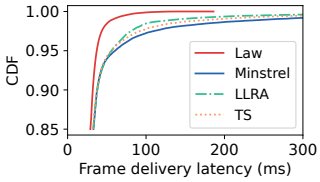
- *Minstrel* [8] is the default RAA used by Linux systems and is normally considered as the state-of-the-art reactive RAA.
- *LLRA* [55] puts latency into the decision-making rationales. But it is also only an algorithmic level optimization, with no innovations in structure and control granularity.
- *TS* [51] utilizes Thompson-sampling to select data rate. Thompson-sampling is a machine learning algorithm for online decision problems under uncertainty where actions are taken sequentially. TS is considered as the state-of-the-art reactive RAA with machine learning.

4.2 Packet-level Performance

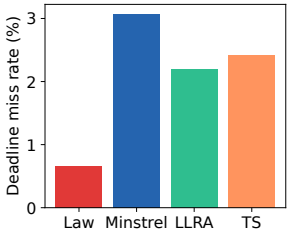
Latency improvement. We validate the bounded latency for each packet provided by Law and compare the per-packet latency with baselines. We present the CDF of per-packet latency in Fig. 16a. Law's curve shows a clear cutoff around 60 milliseconds. After counting the slight latency from the router-to-OpenWiFi wired transmission and the kernel, we can consider this result as a reasonable proof that the latency bound at the link layer is in effect. Compared to three



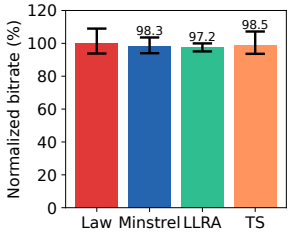
(a) Frame delivery latency.



(b) Latency CDF of non-retransmitted frame.



(c) Deadline miss rate.



(d) Average bitrate.

Figure 17: The performance of Law and three baselines on frame-level. Law shows significant improvement regarding 99ile, 99.9ile latency, and deadline miss rate. Besides, Law provides the highest average bitrate.

baselines (Minstrel, LLRA, and TS), Law reduces 99ile per-packet latency by 47.9% to 78.7%, and 99.9ile per-packet latency by 75.3% to 83.2%.

Loss control. We also measured Law's performance on packet loss control to ensure that Law utilizes a reasonable range of packet loss to be theoretically compatible with the application layer. We take the latency bound set in Law as the deadline for each packet. We show the average loss rate and deadline miss rate in Fig. 16b. The average loss rate of Law is slightly lower than 1%, which is illustrated to be well within the application layer's recovery capabilities in §2.3.

Besides, Law has the best result in terms of the sum of loss rate and deadline miss rate. Even as we discussed in §3.1 and §3.3, due to the trade-off for maximized transmission efficiency, it is hard to achieve the ideal strategy that only drops packets with latency higher than the overall latency bound. Law still achieves the best in this metric, which is close to the optimal performance. We further tested the frequency of consecutive losses, i.e., the average number of five consecutive packet losses per second. We present the results in Fig. 16c. Even with a higher overall loss rate, we reduce the frequency of consecutive losses by 16.3% to 56.3%, demonstrating that Law is able to control the loss pattern well to make it more suitable for low-latency streaming applications.

4.3 Application-level Performance

We then evaluate Law performance with real-world low-latency streaming to see if Law actually shows improvement in real-world applications.

Comparison with baselines. We first measure the per-frame delivery latency and plot the result in Fig. 17a. Law reduces 99ile frame delivery latency by 50.6% to 70.3%, and reduces 99.9ile frame delivery latency by at least 58.4%. We also measure the average deadline miss rate, i.e., the ratio of frames experiencing a latency of more than 100ms. 100ms is a frequently used deadline and requirement for ultra-low-latency streaming [37, 48, 61]. We can see the results in Fig. 17c and Law reduces the deadline miss rate by 69.8% to 78.5%.

In addition, we show the role of Law by calculating the latency of non-retransmitted frames, which refers to frames in which the last received packet was either recovered by the FEC or transmitted for the first time. We picture the latency CDF of this group of frames in Fig. 17b. It is clear that Law shows a cutoff around 180ms while all the baselines have non-retransmitted frames with more than 300ms latency. Note that due to the keyframes and pacing, it is normal in a 30 fps streaming for large frames to take dozens of milliseconds just to complete sending. Coupled with slight latency fluctuations in the rest of the transmission system, it is reasonable to believe that latency bound in the link layer imposes an effective impact.

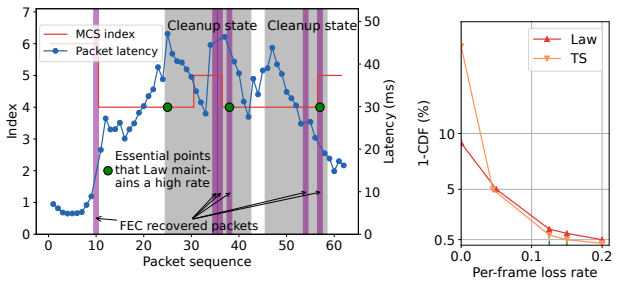
Finally, we measure the effect of Law on video bitrate. And we can tell from Fig. 17d, Law could achieve a slightly higher bitrate than all three baselines, showing that Law is able to promote more aggressive bitrate decisions while ensuring stable low latency.

In-depth exploration. We further present a trace in Fig. 18a consists of three frames to show the underlying rationale for the improvement our design delivers. Firstly, it can be told that we dropped six packets in three frames, a nearly 10% packet loss rate. Five of them are dropped in the Cleanup state. But these packets are well within FEC's recovery capability. So Law perfectly distributes the dropped packets by retransmission control and avoids blockage in the link layer without additional overhead.

Secondly, at the time points shown by the three green dots in the figure, Law is able to keep the MCS at least 4, even with the degradation of channel quality and the increase in retransmissions. Thus, the packet latency is stabilized by fast sending, and the rate can even be adjusted upwards to further clear the waiting packets. With all these accurate controls, these frames are all delivered within the 100 ms deadline.

We also present the comparison of per-frame loss rate in Fig. 18b between Law and TS to justify our observations. Law has almost doubled frames with at least one lost packet. However, Law reduces 65.3% in terms of the percentage of frames with more than 20% packet loss.

Comparison with commercial routers. Besides these RAA baselines above, we also choose some commercial firmwares for comparison. However, these commercial APs typically support 40/80 MHz bandwidth, multiple-in-multiple-out (MIMO), and AC/AX data rate, which means they origi-



(a) A trace that shows Law is able to keep sending fast and wisely drop packets to maintain low latency.

(b) CDF of per-frame loss rate of Law and TS.

Figure 18: An in-depth look at the underlying reasons why Law can lead to improvements.

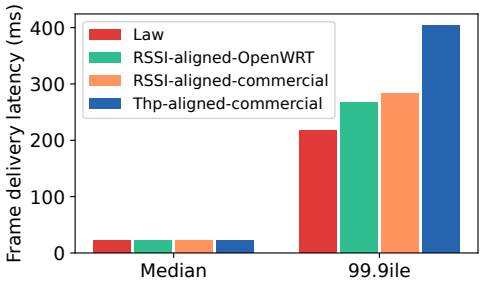


Figure 19: Median and 99.9ile frame delivery latency of Law and commercial routers.

nally provide higher throughput. So, we align some settings and try our best to achieve a relatively fair comparison:

- *RSSI-aligned-OpenWRT:* We use TP LINK WDR4310 refreshed with OpenWRT 23.05.0 for this baseline, setting the Wi-Fi version and bandwidth to 802.11N and 20 MHz correspondingly. We further adjust the tx power and use the same antennas to ensure the RSSI of the OpenWRT AP and the Law is the same. However, TL-WDR4310 supports MIMO2X2 and can not be disabled, so it theoretically provides a doubled maximum throughput.
- *RSSI-aligned-commercial:* We directly use Edimax BR6208 AC750 for this baseline. We align the RSSI in the same way as mentioned above. BR6208AC [5] does not support MIMO for 5G, but the version of the 802.11 protocol is fixed to AC, so it still provides slightly higher theoretical throughput than Law.
- *Thp-aligned-commercial:* We also use Edimax BR6208 for this baseline. We adjust the position of the AP and ensure an equal throughput (Thp) as Law for BBR flows.

We present the frame delivery latency in Fig. 19. Law could provide almost the same median latency compared to all three baselines. Moreover, even in these slightly unfair tests, we can still reduce the 99.9ile tail latency by 18.6% and 22.9% compared with two RSSI-aligned baselines. As for the Thp-aligned baseline, Law reduces the 99.9ile tail latency by 46.0%.

4.4 Microbenchmarking

We also benchmark Law for realistic performance concerns from various aspects.

Parameter setting. We first measure the impact of changing the latency bound. We set the $Limit_q$ to 54ms, which means the total latency bound is changed to 60ms. And we present the difference of frame delivery latency between these two settings in Fig. 20. We can tell that Law with 60ms bound has 3.6% lower 99ile latency while Law with 46ms (default) bound has 15.1% lower 99.9ile latency. But Law with 60ms bound still reduces 99.9ile frame delivery latency by at least 50.9% compared to baselines. Generally, Law can achieve relative latency performance under different settings and. However, a reasonably lower latency bound provides a better reduction in extreme tail latency.

Handling competing flow. We further test the impact of additional competing flow on the performance of Law. We add a 2 Mbps UDP competing flow using Iperf. The results are shown in the Fig. 21. Law still shows great improvement compared to the RSSI-aligned-OpenWRT with 25.4% and 21.3% lower latency at the 99ile and 99.9ile. Law still provides significant performance improvement even with bandwidth contention.

Performance against TCP BBR. We also test how TCP BBR flows react to Law to see Law's compatibility with TCP. We calculate the average throughput of BBR flows for Law and three baselines in Fig. 22. And Law performs similarly with all the baselines, with at least 0.7 Mbps increase compared to Minstrel and LLRA, and 0.7 Mbps reduction compared to TS. This proves that Law works well with TCP BBR flows, not only low-latency streaming applications.

Fairness. We finally measure Law's impact to fairness. We evaluate the performance of Law under both TCP BBR and GCC low-latency streaming, and calculate the Jain Fairness Index. The comparison of Law and Minstrel is shown in Fig. 23. Law also provides decent fairness for both BBR and GCC flows. For TCP BBR, Law leads to a reduction of less than 1% in the median, but also brings a small increase in the lower bound and is generally more stable. In the case of GCC, Law shows a better fairness than Minstrel.

5 Discussion

In this section, we discuss some potential limitations and future work of Law.

Differences and coexistence with AQM. AQM algorithms [17] are a family of queue management algorithms used in network devices to reduce network latency while keeping throughput as high as possible [32, 33, 64, 66]. As a matter of fact, CoDel-FQ has been adopted by the Linux Wi-Fi subsystem in flow queues. Our work is completely different from and fully orthogonal to AQM. AQM often operates

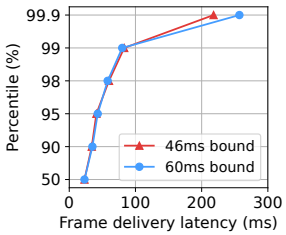


Figure 20: Impact of latency bound. Law maintains a similar performance under different settings.

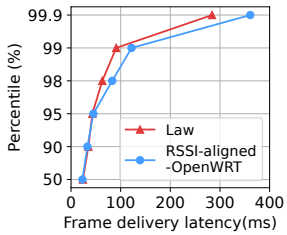


Figure 21: Impact of competing flow. Law still outperforms the commercial router.

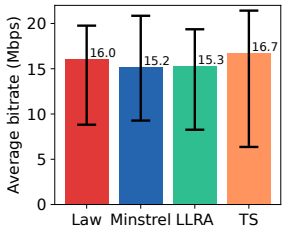


Figure 22: Average throughput of Law and baselines for TCP BBR. Law could provide comparable throughput.

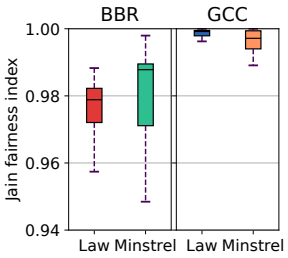


Figure 23: Fairness comparison between Law and Minstrel. Law does not harm transmission fairness.

within a single layer of queues and does not pay attention to queueing structure, or link layer data rate and retransmission. We modified the queueing structure and proposed a new rate and retransmission control algorithm. However, AQM can still be deployed at the layer of queues in the driver that we retained. Our action of setting a latency bound does not conflict with AQM, we can simply treat it as an additional condition of the active discard action in the AQM algorithm. We leave implementing AQM into Law as our future work.

Differentiating loss-sensitive flows. As a latency-aware Wi-Fi link layer design, we demonstrate Law's improvement for tail latency in low-latency streaming and performance maintenance for TCP traffic with latency-sensitive CCAs. There are still many network services that use loss-sensitive CCAs, such as Reno and Cubic. The most straightforward way to differentiate latency-sensitive traffic is to use DSCP bits or even simply maintain an IP whitelist. Some recent solutions are already using DSCP for differentiating flows [18, 71]. However, these methods tend to lack scalability, and the most intelligent approach is still to identify flows based on their characteristics. The major difference between these two types of traffic is that latency-sensitive applications keep the network buffer low, while loss-sensitive traffic tries to fill the buffer first. We can easily distinguish them from each other, and related flow detection work has been proposed [57, 59]. After differentiating loss-sensitive flows, we can switch back to the original transmission schemes for packets in these flows and easily handle these traffics. We leave differentiating loss-sensitive flows as our future work.

6 Related Work

Loss recovery. Packet loss recovery has been a hot topic in tail-latency optimization for low-latency streaming. Since network latency is still one of the bottlenecks of low-latency streaming, packet loss recovery could avoid retransmission and directly reduce tail latency. There are many previous research efforts in optimizing the FEC redundancy strategies [23,35,46,50,69]. Also, some works proposed joint optimization of retransmission and redundancy [31,61,80]. Besides, there are research efforts trying to utilize loss-tolerant codecs to recover lost data [25, 36, 49, 52, 74, 77]. However, they are not enough to solve the problem of long tail latency. As we discussed in §2.2, the loss rate is actually very low in existing Wi-Fi networks even when latency increases dramatically. This means that recovery abilities in the upper layer are not fully utilized, and link layer structure optimization is still critically required.

Wi-Fi latency optimization. Nowadays, there is also some work on latency optimization for Wi-Fi networks from different aspects. Apart from WMM, AQM, and latency-aware RAA [55] mentioned above, there are also optimizations by flow scheduling. Some flow scheduling aims to improve airtime fairness [38, 39, 43], and some are specially proposed to achieve a better performance for video applications [57, 65]. There are also studies on frame aggregation strategies [41, 58, 70] to achieve better latency and transmission overhead. Link layer CCAs [30, 67] have also been proposed to reduce transmission latency in a heavily congested channel. However, these algorithm-level designs are far from sufficient to address the high tail latency in Wi-Fi last-hop under the existing link layer structure. Law could significantly improve latency performance mainly due to: (i) architectural optimization in Wi-Fi link layer towards consistent low latency, and (ii) a fine-grained latency-sensitive rate and retransmission control.

7 Conclusion

This paper proposes Law, an 802.11 link layer structure towards consistent low latency for ultra-low-latency applications. By reshaping the queueing structure and designing fine-grained data rate and retransmission control, Law is able to utilize the loss-tolerance capacity from the application layer and significantly reduce the tail latency at both the packet and application levels while maintaining an acceptable packet loss. Our code is now available at https://github.com/hkust-spark/Law-NSDI-26.

Acknowledgements. We thank our shepherd, Hongqiang Liu, and the anonymous NSDI reviewers for valuable comments. This work is supported by RGC Early Career Scheme (26212525) and National Key R&D Program of China (2025YFE0201000). Zili Meng is the corresponding author.

References

[1] 802.11e - wi-fi multimedia (wmm). https://www.rhyshaden.com/wifi_mm.htm.

[2] Ad9631 datasheet and product info. https://www.analog.com/en/products/ad9631.html.

[3] Amd zynq 7000 socs. https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-7000.html.

[4] Code repository for openwifi. https://github.com/open-sdr.

[5] Edimax br6208ac. https://www.edimax.com/edimax/merchandise/merchandise_detail/data/edimax/global/home_legacy_wireless_routers/br-6208ac_v2/.

[6] Linux wireless documentation for ath9k. https://wireless.docs.kernel.org/en/latest/en/users/drivers/ath9k.html.

[7] Paced sending. https://webrtc.googlesource.com/src/+/refs/heads/main/modules/pacing/g3doc/index.md.

[8] Rate adaptation for 802.11 wireless networks: Minstrel. https://blog.cerowrt.org/papers/minstrel-sigcomm-final.pdf.

[9] Retransmission limit in ath9k. https://elixir.bootlin.com/linux/v6.15-rc3/source/drivers/net/wireless/ath/ath9k/xmit.c.

[10] Tp-link tl-wdr4310. https://oldwiki.archive.openwrt.org/toh/tp-link/tl-wdr4310.

[11] 802.11 qos tutorial. https://www.ieee802.org/1/files/public/docs2008/avb-gs-802-11-qos-tutorial-1108.pdf, 2008.

[12] Psa: Webrtc m88 release notes. https://groups.google.com/g/discuss-webrtc/c/A0FjOcTW2c0/m/UAv-veyPCAAJ, 2020.

[13] Azure network round-trip latency statistics. https://learn.microsoft.com/en-us/azure/networking/azure-network-latency, 2024.

[14] Speedtest global index. https://www.speedtest.net/global-index, 2025.

[15] Dmitry Akhmetov, Dibakar Das, Dave Cavalcanti, Javier Ramirez-Perez, and Laurent Cariou. Scheduled time-sensitive transmission opportunities over wi-fi. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, pages 1807–1812, 2022.

[16] Venkat Arun and Hari Balakrishnan. Copa: Practical Delay-Based congestion control for the internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 329–342, Renton, WA, April 2018. USENIX Association.

[17] F. Baker and G. Fairhurst. Rfc 7567: Ietf recommendations regarding active queue management, 2015.

[18] Fred Baker, Jozef Babiarz, and Kwok Ho Chan. Configuration Guidelines for DiffServ Service Classes. RFC 4594, August 2006.

[19] Apurv Bhartia, Bo Chen, Feng Wang, Derrick Pallas, Raluca Musaloiu-E, Ted Tsung-Te Lai, and Hao Ma. Measurement-based, practical techniques to improve 802.11ac performance. In *Proceedings of the 2017 Internet Measurement Conference*, IMC '17, page 205–219, New York, NY, USA, 2017. Association for Computing Machinery.

[20] Saad Biaz and Shaoen Wu. Rate adaptation algorithms for ieee 802.11 networks: A survey and comparison. In *2008 IEEE Symposium on Computers and Communications*, pages 130–136, 2008.

[21] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *ACM Queue*, 14, September-October:20 – 53, 2016.

[22] Batyr Charyyev, Engin Arslan, and Mehmet Hadi Gunes. Latency comparison of cloud datacenters and edge servers. In *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pages 1–6, 2020.

[23] Ke Chen, Han Wang, Shuwen Fang, Xiaotian Li, Minghao Ye, and H Jonathan Chao. Rl-afec: adaptive forward error correction for real-time video communication based on reinforcement learning. In *Proceedings of the 13th ACM Multimedia Systems Conference*, pages 96–108, 2022.

[24] Syuan-Cheng Chen, Chi-Yu Li, and Chui-Hao Chiu. An experience driven design for ieee 802.11ac rate adaptation based on reinforcement learning. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, 2021.

[25] Yihua Cheng, Ziyi Zhang, Hanchen Li, Anton Arapin, Yue Zhang, Qizheng Zhang, Yuhan Liu, Kuntai Du, Xu Zhang, Francis Y Yan, et al. {GRACE}:{Loss-Resilient}{Real-Time} video through neural codecs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 509–531, 2024.

[26] Yousri Daldoul, Djamal-Eddine Meddour, and Adlen Ksentini. Performance evaluation of ofdma and mu-mimo in 802.11ax networks. *Computer Networks, Volume 182, 9 December 2020*, 2020. Elsevier. Personal use of this material is permitted. The definitive version of this paper was published in Computer Networks, Volume 182, 9 December 2020 and is available at : https://doi.org/10.1016/j.comnet.2020.107477.

[27] The Khang Dang, Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Jörg Ott, and Jussi Kangasharju. Cloudy with a chance of short rtts: Analyzing cloud connectivity in the internet. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 62–79, 2021.

[28] Luca De Cicco, Gaetano Carlucci, and Saverio Mascolo. Understanding the dynamic behaviour of the google congestion control for rtcweb. In *2013 20th International Packet Video Workshop*, pages 1–8, 2013.

[29] Sandesh Dhawaskar Sathyanarayana, Kyunghan Lee, Dirk Grunwald, and Sangtae Ha. Converge: Qoe-driven multipath video conferencing over webrtc. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 637–653, New York, NY, USA, 2023. Association for Computing Machinery.

[30] Xinle Du, Jie Li, Yiyang Shao, Wei Wang, Shuihai Hu, Jingbin Zhou, and Kun Tan. Revisiting congestion control for wifi networks. In *Proceedings of the 8th Asia-Pacific Workshop on Networking*, pages 88–94, 2024.

[31] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. Reducing web latency: the virtue of gentle aggression. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 159–170, 2013.

[32] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[33] Sally Floyd, Ramakrishna Gummadi, Scott Shenker, et al. Adaptive red: An algorithm for increasing the robustness of red's active queue management, 2001.

[34] Silas L. Fong, Ashish Khisti, Baochun Li, Wai-Tian Tan, Xiaoqing Zhu, and John Apostolopoulos. Optimal streaming codes for channels with burst and arbitrary erasures. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 1370–1374, 2018.

[35] Silas L Fong, Ashish Khisti, Baochun Li, Wai-Tian Tan, Xiaoqing Zhu, and John Apostolopoulos. Optimal streaming codes for channels with burst and arbitrary erasures. *IEEE Transactions on Information Theory*, 65(7):4274–4292, 2019.

[36] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. Salsify:{Low-Latency} network video through tighter integration between a video codec and a transport protocol. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 267–282, 2018.

[37] Lorenzo Galati-Giordano, Giovanni Geraci, Marc Carrascosa, and Boris Bellalta. What will wi-fi 8 be? a primer on ieee 802.11bn ultra high reliability. *IEEE Communications Magazine*, 62(8):126–132, 2024.

[38] Rosario G Garroppo, Stefano Giordano, Stefano Lucetti, and Luca Tavanti. Providing air-time usage fairness in ieee 802.11 networks with the deficit transmission time (dtt) scheduler. *Wireless Networks*, 13:481–495, 2007.

[39] Karina Gomez, Roberto Riggio, Tinku Rasheed, and Imrich Chlamtac. On efficient airtime-based fair link scheduling in ieee 802.11-based wireless networks. In *2011 IEEE 22nd International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 930–934. IEEE, 2011.

[40] Prateesh Goyal, Anup Agarwal, Ravi Netravali, Mohammad Alizadeh, and Hari Balakrishnan. ABC: A simple explicit congestion controller for wireless networks. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 353–372, Santa Clara, CA, February 2020. USENIX Association.

[41] Nasreddine Hajlaoui, Issam Jabri, Malek Taieb, and Maher Benjemaa. A frame aggregation scheduler for qos-sensitive applications in ieee 802.11 n wlans. In *2012 international conference on communications and information technology (ICCIT)*, pages 221–226. IEEE, 2012.

[42] Daniel Halperin, Wenjun Hu, Anmol Sheth, and David Wetherall. Predictable 802.11 packet delivery from wireless channel measurements. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, page 159–170, New York, NY, USA, 2010. Association for Computing Machinery.

[43] Toke Høiland-Jørgensen, Michał Kazior, Dave Täht, Per Hurtig, and Anna Brunstrom. Ending the anomaly: Achieving low latency and airtime fairness in {WiFi}. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 139–151, 2017.

[44] Stefan Holmer, Henrik Lundin, Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. A Google Congestion Control Algorithm for Real-Time Communication. Internet-Draft draft-alvestrand-rmcat-congestion-03, Internet Engineering Task Force, June 2015. Work in Progress.

[45] Stefan Holmer, Mikhal Shemer, and Marco Paniconi. Handling packet loss in webrtc. In *2013 IEEE International Conference on Image Processing*, pages 1860–1864, 2013.

[46] Han Hu, Sheng Cheng, Xinggong Zhang, and Zongming Guo. Lightfec: Network adaptive fec with a lightweight deep-learning approach. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 3592–3600, 2021.

[47] Xianjun Jiao, Wei Liu, Michael Mehari, Muhammad Aslam, and Ingrid Moerman. openwifi: a free and open-source ieee802. 11 sdr implementation on soc. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–2. IEEE, 2020.

[48] Teemu Kämäräinen, Matti Siekkinen, Antti Ylä-Jääski, Wenxiao Zhang, and Pan Hui. A measurement study on achieving imperceptible latency in mobile cloud gaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, MMSys'17, page 88–99, New York, NY, USA, 2017. Association for Computing Machinery.

[49] Vineeth Kolkeri, Muhammad Koul, J Lee, and K.R. Rao. Error concealment techniques in h. 264/avc for wireless video transmission in mobile networks. 01 2008.

[50] Balázs Kreith, Varun Singh, and Jörg Ott. Fractal: Fec-based rate control for rtp. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 1363–1371, 2017.

[51] Alexander Krotov, Anton Kiryanov, and Evgeny Khorov. Rate control with spatial reuse for wi-fi 6 dense deployments. *IEEE Access*, 8:168898–168909, 2020.

[52] Sunil Kumar, Liyang Xu, Mrinal Mandal, and Sethuraman Panchanathan. Error resiliency schemes in h.264/avc standard. *Journal of Visual Communication and Image Representation*, 17:425–450, 04 2006.

[53] Insoo Lee, Seyeon Kim, Sandesh Sathyanarayana, Kyungmin Bin, Song Chong, Kyunghan Lee, Dirk Grunwald, and Sangtae Ha. R-fec: Rl-based fec adjustment for better qoe in webrtc. In *Proceedings of the 30th ACM International Conference on Multimedia*, MM '22, page 2948–2956, New York, NY, USA, 2022. Association for Computing Machinery.

[54] Chi-Yu Li, Chunyi Peng, Songwu Lu, and Xinbing Wang. Energy-based rate adaptation for 802.11n. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Mobicom '12, page 341–352, New York, NY, USA, 2012. Association for Computing Machinery.

[55] Chi-Yu Li, Chunyi Peng, Songwu Lu, Xinbing Wang, and Ranveer Chandra. Latency-aware rate adaptation in 802.11 n home networks. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 1293–1301. IEEE, 2015.

[56] Wei Li and Zili Meng. Poster: Content-aware retransmission for ultra-low-latency video streaming. In *Proceedings of the ACM SIGCOMM 2024 Conference: Posters and Demos*, pages 66–68, 2024.

[57] Shao-Jung Lu, Wei-Xun Chen, Yu-Shao Su, Yu-Shou Chang, Yao-Wen Liu, Chi-Yu Li, and Guan-Hua Tu. Practical latency-aware scheduling for low-latency elephant vr flows in wi-fi networks. In *2024 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 57–68, 2024.

[58] Bakeel Maqhat, Mohd Dani Baba, and Ruhani Ab Rahman. A-msdu real time traffic scheduler for ieee802. 11n wlans. In *2012 IEEE Symposium on Wireless Technology and Applications (ISWTA)*, pages 286–290. IEEE, 2012.

[59] Zili Meng, Nirav Atre, Mingwei Xu, Justine Sherry, and Maria Apostolaki. Confucius: Achieving consistent low latency with practical queue management for real-time communications, 2024.

[60] Zili Meng, Yaning Guo, Chen Sun, Bo Wang, Justine Sherry, Hongqiang Harry Liu, and Mingwei Xu. Achieving consistent low latency for wireless real-time communications with the shortest control loop. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 193–206, 2022.

[61] Zili Meng, Xiao Kong, Jing Chen, Bo Wang, Mingwei Xu, Rui Han, Honghao Liu, Venkat Arun, Hongxin Hu, and Xue Wei. Hairpin: Rethinking packet loss recovery in edge-based interactive video streaming. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 907–926, Santa Clara, CA, April 2024. USENIX Association.

[62] Zili Meng, Tingfeng Wang, Yixin Shen, Bo Wang, Mingwei Xu, Rui Han, Honghao Liu, Venkat Arun,

Hongxin Hu, and Xue Wei. Enabling high quality {Real-Time} communications with adaptive {Frame-Rate}. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1429–1450, 2023.

[63] Gaurang Naik, Dennis Ogbe, and Jung-Min Jerry Park. Can wi-fi 7 support real-time applications? on the impact of multi link aggregation on latency. In *ICC 2021 - IEEE International Conference on Communications*, pages 1–6, 2021.

[64] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar. Rfc 8289: Controlled delay active queue management, 2018.

[65] Summera Nosheen and Jamil Y Khan. High definition video packet scheduling algorithms for ieee802. 11ac networks to enhance qoe. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–5. IEEE, 2020.

[66] Rong Pan, Preethi Natarajan, Chiara Piglione, Mythili Suryanarayana Prabhu, Vijay Subramanian, Fred Baker, and Bill VerSteeg. Pie: A lightweight control scheme to address the bufferbloat problem. In *2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*, pages 148–155, 2013.

[67] Changhua Pei, Youjian Zhao, Yunxin Liu, Kun Tan, Jiansong Zhang, Yuan Meng, and Dan Pei. Latency-based wifi congestion control in the air for dense wifi networks. In *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, pages 1–10, 2017.

[68] Devdeep Ray, Connor Smith, Teng Wei, David Chu, and Srinivasan Seshan. Sqp: Congestion control for low-latency interactive video streaming, 2022.

[69] Michael Rudow, Francis Y Yan, Abhishek Kumar, Ganesh Ananthanarayanan, Martin Ellis, and KV Rashmi. Tambur: Efficient loss recovery for video-conferencing via streaming codes. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 953–971, 2023.

[70] Anwar Saif and Mohamed Othman. Sra-msdu: Enhanced a-msdu frame aggregation with selective retransmission in 802.11 n wireless networks. *Journal of Network and Computer Applications*, 36(4):1219–1229, 2013.

[71] Koen De Schepper, Bob Briscoe, and Greg White. Dual-Queue Coupled Active Queue Management (AQM) for Low Latency, Low Loss, and Scalable Throughput (L4S). RFC 9332, January 2023.

[72] Marie-Theres Suer, Prince Jose, and Hugues Tchouankem. Experimental evaluation of ieee 802.11ax - low latency and high reliability with wi-fi 6? In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, pages 377–382, 2022.

[73] Shibo Wang, Shusen Yang, Xiao Kong, Chenglei Wu, Longwei Jiang, Chenren Xu, Cong Zhao, Xuesong Yang, Jianjun Xiao, Xin Liu, Changxi Zheng, Jing Wang, and Honghao Liu. Pudica: Toward Near-Zero queuing delay in congestion control for cloud gaming. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 113–129, Santa Clara, CA, April 2024. USENIX Association.

[74] Yi Wang, Xiaoqiang Guo, Ye Feng, Aidong Men, and Bo Yang. A novel temporal error concealment framework in h.264/avc. *Proceedings - IEEE International Conference on Multimedia and Expo*, pages 1–6, 07 2013.

[75] Stephen B. Wicker and Vijay K. Bhargava. *Reed-Solomon Codes and Their Applications*. John Wiley & Sons, Inc., USA, 1999.

[76] Starsky H. Y. Wong, Hao Yang, Songwu Lu, and Vaduvur Bharghavan. Robust rate adaptation for 802.11 wireless networks. In *Proceedings of the 12th Annual International Conference on Mobile Computing and Networking*, MobiCom '06, page 146–157, New York, NY, USA, 2006. Association for Computing Machinery.

[77] Jiangkai Wu, Yu Guan, Qi Mao, Yong Cui, Zongming Guo, and Xinggong Zhang. Zgaming: Zero-latency 3d cloud gaming by image prediction. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 710–723, 2023.

[78] Hao Yin, Murali Ramanujam, Joe Schaefer, Stan Adermann, Srihari Narlanka, Perry Lea, Ravi Netravali, and Krishna Chintalapudi. ADR-X: ANN-Assisted wireless link rate adaptation for Compute-Constrained embedded gaming devices. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1331–1349, Santa Clara, CA, April 2024. USENIX Association.

[79] Mo Zanaty, Varun Singh, Ali C. Begen, and Giridhar Mandyam. RTP Payload Format for Flexible Forward Error Correction (FEC). RFC 8627, July 2019.

[80] Fan Zhai, Yiftach Eisenberg, Thrasyvoulos N Pappas, Randall Berry, and Aggelos K Katsaggelos. Rate-distortion optimized hybrid error control for real-time packetized video transmission. *IEEE Transactions on Image Processing*, 15(1):40–53, 2006.

[81] Zhilong Zheng, Yunfei Ma, Yanmei Liu, Furong Yang, Zhenyu Li, Yuanbo Zhang, Jiuhai Zhang, Wei Shi, Wentao Chen, Ding Li, Qing An, Hai Hong, Hongqiang Harry Liu, and Ming Zhang. Xlink: Qoe-driven multi-path quic transport in large-scale video services. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 418–432, New York, NY, USA, 2021. Association for Computing Machinery.

[82] Yuhan Zhou, Tingfeng Wang, Liying Wang, Nian Wen, Rui Han, Jing Wang, Chenglei Wu, Jiafeng Chen, Longwei Jiang, Shibo Wang, Honghao Liu, and Chenren Xu. AUGUR: Practical mobile multipath transport service for low tail latency in Real-Time streaming. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1901–1916, Santa Clara, CA, April 2024. USENIX Association.

Appendices

Appendices are supporting material that has not been peer-reviewed.

A Compatibility with Commercial Hardware

As mentioned in the main text, we retain the original control process unchanged in data processing, except for the queueing structure, retransmission and rate control. This also indicates that although we have modified the queueing structure, it does not mean that Law is completely in conflict with the original queues. Law can be treated as a patch or submodule for the current link layer design. All packets in the router will be divided into flows, so one possible solution is to deploy parts of Law above the hardware as a standby submodule. Only latency-sensitive flows are directed to the Law, while the remaining flows continue to pass through the original queues. Packets from both are rejoined when entering the hardware, which uses the Law hardware design. Another solution is to completely replace the queueing structure and switch data rate and retransmission control algorithms to meet the requirements of different applications. Different approaches face tradeoffs between system overhead, throughput, fairness, latency, and so on.

B Parameter Setting

Here, we introduce how we choose the default parameter settings, such as $Limit_q$ and the hardware retransmission limit. The value of $Limit_q$ can be ideally determined based on application throughput, latency requirements, and the level of loss tolerance. Taking our experimental setup as an example, we want to control the packet loss rate below $(1-\alpha)$. We can find a latency threshold such that, under a certain baseline, the proportion of packets has a latency lower than this threshold is exactly α. In this way, we can obtain multiple threshold values β_n from different baselines, as shown in Fig. 24. We can then select a value within the interval from β_1 to β_2 as the overall latency bound, and try to set $Limit_q$ based on this.

We also make the following explanations to discuss some of our observations during implementation and clarify why we set the hardware retransmission limit as given in §3.4. If we ideally treat each hardware transmission as an independent random event with a success probability of p, then the probability of a packet being successfully received just at the n^{th} hardware transmission can be calculated as:

$$X_n = p*(1-p)^{n-1} \quad (n = 1,2,3,\ldots) \tag{3}$$

When $p > 0.5$, we can easily prove that:

$$X_n > X_{n+1} + X_{n+2} + X_{n+3} + \ldots \tag{4}$$

This indicates that the more retransmissions we have, Law's efficiency in the tradeoff between latency (number of retransmissions) and transmission success rate decreases. Furthermore, the larger p is, the smaller the gains in the transmission success rate from massive retransmissions.

Even if the moving average probability of successful hardware transmission for a certain data rate is p, the probability of successful transmission in the extremely short time (a short interval under specific channel conditions) is not p. For example, from our observation, if the chosen data rate handles the channel conditions well, the probability of successful transmission will often be closer to 1 than p. This leads to a more unworthy tradeoff, as we just discussed. That's the reason we set the hardware-related parameters as in §3.4, enabling a more sensitive and more accurate rate adaptation with an efficient latency-reliability tradeoff.

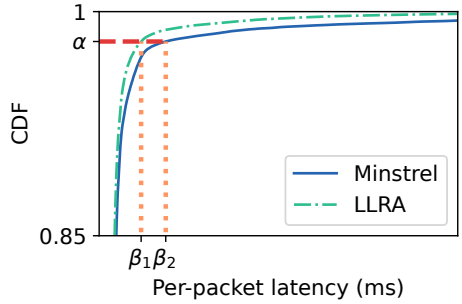


Figure 24: Illustration for the threshold interval calculation.