Latency-Aware Rate Adaptation in 802.11n Home Networks

Chi-Yu Li*, Chunyi Peng†, Songwu Lu*, Xinbing Wang‡ and Ranveer Chandra§
*University of California, Los Angeles, USA †The Ohio State University, USA
‡Shanghai Jiao Tong University, China §Microsoft Research, USA
*{lichiyu, slu}@cs.ucla.edu, †chunyi@cse.ohio-state.edu, ‡xwang8@sjtu.edu.cn, §ranveer@microsoft.com

***Abstract*—Latency-sensitive applications (e.g., wireless gaming and TV remote play) are increasingly popular in home WiFi networks. Such millisecond-level latency requirements call for new fine-grained approaches at the link layer. In this paper, we show that current solutions work well for throughput but not for latency due to the long tail of the packet delay distribution. We thus propose LLRA, a new latency-aware rate adaptation scheme that reduces the tail latency for delay-sensitive applications. LLRA takes concerted design in rate control, frame aggregation scheduling and software/hardware retransmission dispatching. Our implementation and evaluation confirm the viability of LLRA in 802.11n home networks.**

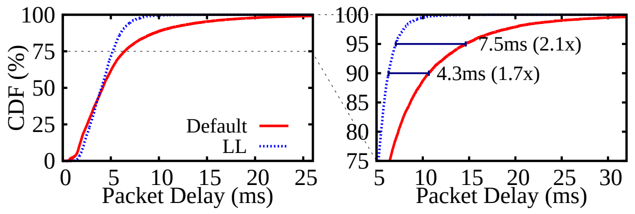


Figure 1: The cumulative distribution function (CDF) of packet delay for the default setting and the low-latency (LL) one at the observed client in an 802.11n home network.

I. Introduction

Wi-Fi has been widely deployed in home networking environment. Over 450 million households worldwide currently have Wi-Fi networks set up in 2014, reaching 65% penetration of fixed-line broadband households; by 2018, WiFi access at home is projected to almost double, covering 800 million households, according to the recent report by Strategy Analytics [8]. In a typical home WiFi deployment scenario today, an 802.11n[1] AP is used to serve multiple WiFi-capable devices, including laptops, tablets, smartphones, game consoles and other gadgets. Home users not only use them for the Internet access, but also play games, access high-fidelity music, watch online videos on TV, and to name a few.

We focus on serving latency-sensitive data flows that quest for millisecond-level delays in a home WiFi network. Each flow denotes a stream of packets from a source to a destination. The per packet delay is the duration from the time a packet arrives at the link layer to the time its acknowledgement (ACK) is received. For each latency-sensitive flow, we seek to improve the long tail of its latency distribution. Our specific goal is to reduce the *tail latency* of a flow, which is defined as the packet latency at the α percentile (say, $\alpha = 90^{th}$ of all packets) over a window of packet bursts of the flow.

Our work is highly motivated by the increasing popularity of real-time or interactive applications in home WiFi networks. They impose stringent (millisecond-level) latency requirement over a single wireless hop. Such applications include wireless console gaming [4], mobile-to-mobile gaming [16], streaming gaming [3], screen mirroring and TV remote play (*e.g.*, via Chromecast [2]), *etc.*. For wireless gaming, timely delivery of commands (from the consoles, phones or other peripherals) and multimedia data if applicable, is critical to the liveness and interactiveness for gaming. It thus requires the latency within several milliseconds or even 1 ms [4]. So is the screen mirroring which projects what we see on the phone or tablet onto TV. The user experience of these applications is sensitive to the tail of the latency distribution (expressed in α percentile) [16]. Outside the home WiFi network, some applications also have low latency requirement over a single wireless hop. For instance, high-fidelity music playback expects a latency within 11.5ms [9], whereas 1-3ms delay is highly desirable in augmented reality [18].

At first glimpse, it seems that current WiFi can already meet the latency requirement. With up to 600Mbps speed at an 802.11n AP, higher data rate implies shorter transmission time, and thereby lower latency. However, our study shows this is not true in general: *Highest speed does not always yield shortest latency*. Figure 1 plots the cumulative distribution functions (CDFs) of packet latency for two rate settings in 802.11n. The default setting achieves the highest speed whereas the other, denoted as low-latency (LL) one, yields shorter per packet delay. In terms of throughput, the former achieves 108.0 Mbps and does outperform the latter (1.2x of 87.4 Mbps by LL). However, it incurs 4–8ms extra delay for the 90^{th} or 95^{th} latency than LL. Specifically, the default setting takes 10.6 ms (90^{th}) and 14.6 ms (95^{th}), compared with 6.3 ms and 7.1ms by LL. The gap of the maximum even reaches up to 38.9 ms (58.3ms versus 19.4ms). The root cause lies in that, highest rate reduces the *average* or medium latency, but not necessarily the tail latency. In the example, both rates contribute to similar medium packet delay (4.0 ms), but the highest rate setting generates a much longer tail, which hurts quality of experience for applications.

Given the fact that the long tail mainly comes from the 802.11 link layer, other higher-layer approaches such as network packet scheduling or transport congestion control manage latency at course grains and are deemed less effective. In this work, we reduce the tail latency from the link rate

[1]It can be 802.11a/b/g/n/ac. 802.11n is a popular example.

adaptation (RA) perspective. Our proposed solution LLRA is a first RA algorithm that achieves millisecond-level latency in a typical, multi-client home 802.11n network. The goal of LLRA is to find a LL rate setting with the lowest latency at the given α percentile. The target tail latency is thus reduced as much as possible. LLRA is the concerted design of three components: rate control, frame aggregation scheduling, and retransmission dispatching. The rate control searches for the LL rate, which balances between initial queue delay and service time, thereby achieving lowest latency. The aggressive aggregation scheduling is further applied to reduce the former. To curtail the latter, the retransmission dispatching prioritizes retransmissions. We take a novel probing-based approach to LL rate search to make it work in real systems. Association rule-based pruning is used to eliminate high-loss rate settings by correlating results at different settings under similar channel conditions. It consequently offers a light-weight solution to protect LLRA from excessive probing cost.

We have implemented LLRA on commodity 802.11n APs. In all tested scenarios, it consistently reduces the tail latency over other algorithms. For the latency at the 90^{th} and 95^{th} percentiles, LLRA reduces tail latency by 30.7%-90.8% and 21.8%-66.2% over ARA [23] and MiRA [19], respectively. For example, while ARA and MiRA observe tail latency of 28.0 ms and 14.4 ms respectively, LLRA keeps it below 10 ms.

The rest of the paper is organized as follows. Section II describes the background and related work. Section III showcases the impacts of RA on latency. Sections IV, V and VI present the design, implementation and evaluation of LLRA. Section VII concludes the work.

II. Background and Related Work

RA is an effective software technique to improve wireless performance, yet left unspecified by the 802.11 standard. In general, RA needs to address three issues: (1) What rate to set given the current condition? This is the task of *rate control* component; (2) Upon what frames to apply rate control? This is the role of *frame aggregation scheduling* component, which may vary the aggregated frame size; (3) How to handle frame failures? This is the function of the *retransmission dispatcher*, which manages retransmission policy. Consequently, each RA has three components that work in concert to serve application flows: (1) rate control; (2) frame aggregation (FA) scheduling; (3) retransmission dispatching. We next elaborate on each.

Rate control for MIMO: MIMO uses multiple transmit and receive antennas. It is able to split a data flow into multiple streams, which are sent through separate transmit antennas. 802.11n supports up to four spatial streams (4 antennas required), offering at most 600 Mbps [6]. The emerging 802.11ac supports 8x8 antenna settings, and boosts speed to 7Gbps [7]. Given an antenna and stream setting, there are several MCS options (varying rates). Rate control adapting to dynamic wireless channels thus is critical to performance.

Frame aggregation (FA): FA packs multiple data packets into an aggregated frame. A Block-ACK is sent if at least one packet in the frame is received without error. Each Block-ACK can acknowledge multiple packets. The maximum size of an aggregation frame is bound by the maximum transmission time of one frame (here, 4 ms), and thus increases with rates.

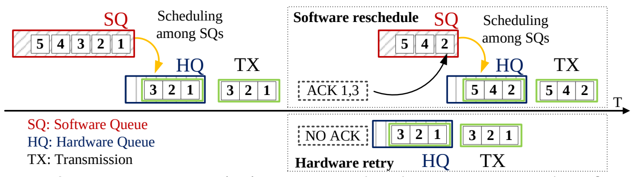


Figure 2: Two retransmission types: hardware retry and software reschedule.

Retransmission: Retransmission is needed to recover from packet losses. There are two types: *hardware retry* and *software reschedule*. Figure 2 shows an example of both mechanisms at an AP. Initially, each packet is placed into the software queue (SQ) once it arrives at the MAC layer. It is not placed to the hardware queue (HQ) until those in other higher-priority SQs (for different clients) are scheduled ahead and transmitted. Once the HQ is available, packets (here, packets 1-3) are aggregated to form an aggregated frame. It is then pushed into the HQ and transmitted over the air. Retransmission for unsuccessful packets depends on whether the ACK is received.

If the ACK is received but not the whole frame succeeds, *software reschedule* is applied to retransmitting unsuccessful packets. A Block-ACK indicates those successful packets and then those failed ones (here, packet 2) are placed back to its original SQ (at the head). Retransmission has to wait for its turn from the scheduling among SQs. The unsuccessful ones may be aggregated with other packets (here, packets 4 and 5). In contrast, if no ACK is received (all aggregated packets fail), *hardware retry* is used. The frame stays in the HQ for retransmission. The maximum number of hardware retries is platform dependent and can be configured. If all the hardware retries fail, *software reschedule* is eventually employed.

Related work: RA has been actively studied in recent years (see [10], [13]–[15], [19], [23] for examples). Most proposals focus on rate control for high goodput in 802.11n or MIMO networks, such as [13], [19], [23], whereas a few optimize energy efficiency [14], [15]. Certain rate-control solution is proposed for specific applications, *e.g.*, encoding-aware probing adaptation helps to reduce latency in video streaming [10]. However, no latency-aware RA has appeared to reduce the long tail, particularly at the millisecond granularity.

Numerous solutions have been proposed to optimize wireless delay, including packet scheduling [11], [24], MAC protocol [17], transport-layer optimization [22], [25], redundancy [21], and application optimization for video streaming [10], [12], [20]. They target the latency that is at least an order of magnitude or more (several hundreds of milliseconds) on mesh networks, ad hoc networks, cellular networks and the Internet. Since millisecond-level tail latency is mainly the product of 802.11 link-layer operations, we believe our RA-based scheme offers a more viable approach. We aim to reduce latency not only via rate control, but also through FA scheduling and retransmission dispatching. Note that our work is orthogonal to 802.11e [5], which prioritizes channel contention and access for different traffic classes.

III. Packet Latency in 802.11n

The latency of a packet consists of three parts:

$$D = T_q + T_{mac} + T_{air}, \tag{1}$$

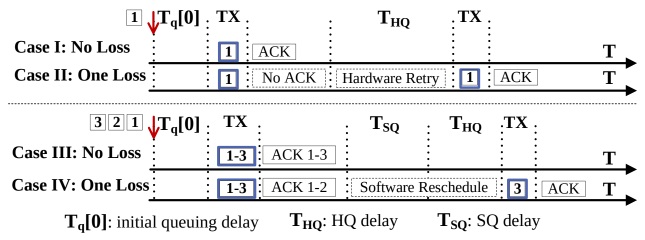


Figure 3: Four illustrative examples of the last packet's latency.

Case	$T_q[0]$	T_{SQ}	T_{HQ}	T_{air}	T_{mac}	L (ms)
I	1	0	0	0.2	0.2	1.4 (100%)
II	1	0	**2**	0.15**+0.15**	0.2**+0.2**	3.7 (264%)
III	1	0	0	0.2*3	0.2	1.8 (100%)
IV	1	**5**	**2**	0.45+**0.15**	0.2**+0.2**	9.0 (500%)

Table I: Latency breakdown of the last packet in four cases.

where T_q is the queuing delay, T_{mac} is the duration for the MAC overhead, and T_{air} is the transmission time on the air.

T_q: the queuing delay includes the initial queuing delay and possible subsequent values due to retransmissions. Its latency comes from two queues: the software queue (T_{SQ}) and the hardware queue (T_{HQ}). T_{SQ} is used to wait for being scheduled to the hardware queue, while T_{HQ} is used to wait for its turn to be delivered once in the HQ. All the packets experience the initial queueing delay, since it counts the waiting period from the packet arrival to its first transmission attempt. For the delay incurred by retransmission, a hardware retry only requires T_{HQ}, whereas a software reschedule includes both, thereby resulting in longer delay.

T_{mac}: It counts the overall MAC overhead for all transmission attempts including retransmissions. For each transmission attempt, the MAC overhead covers the time for contention (backoff), inter-frame space (*e.g.*, DIFS/SIFS) and ACK.

T_{air}: It sums up transmission airtime for all attempts. For a packet in a FA frame, its airtime is determined by the length of the entire frame and the used data rate, since all packets are bundled in their transmission.

A. Illustrative Examples

We now show how packet latency varies with two rate settings in non-FA and FA cases. Figure 3 depicts four illustrative examples. Consider an AP serving multiple clients. For one observed client, assume that the low-rate setting R_L has small packet error rate (PER) so that the initial transmission succeeds. Its per-packet transmission airtime is 0.2 ms. The high rate R_H results in smaller airtime (0.15ms) but larger PER; its first transmission fails but the delivery succeeds after one retransmission. Transmission from other clients of interest takes 2 ms before each transmission of AP. Assume the HQ holds only one frame, so 2 ms can be considered as T_{HQ}. The initial queueing delay is 1 ms. The MAC overhead for each transmission attempt is 0.2 ms. Table I shows the breakdown of the last packet's latency in four cases.

Cases I and II (non-FA cases): As shown in Table I, Packet 1 gets the latency 1.4 ms and 3.7 ms in Cases I and II, respectively. In Case I, it is successfully delivered during the first attempt. However, in Case II, one retransmission via hardware retry is needed, resulting in 2.3 ms extra latency. This shows that PER plays a critical role, yet the airtime saving from higher-goodput rates is almost negligible.

Insight 1: Packet latency is dominated by loss-induced retransmission when other factors have similar effect.

Cases III and IV (FA cases): When FA is used, multiple packets are transmitted together, thus saving MAC overhead. It can be seen that three packets share the MAC overhead in 0.2 ms (Case III). For the last packet (Packet 3), its latency is 1.8 ms and 9.0 ms in both cases. In case IV, retransmission is still the dominant factor, but much longer latency is incurred compared with Case II. This is due to software reschedule, which is triggered when partial aggregation frame succeeds. In such a case, both T_{SQ} and T_{HQ} contribute to the delay. T_{SQ} is obtained by assuming two 2-packet frames from the SQs for other clients prior to this retransmission. Therefore, T_{SQ} is $2 \times 2.5 = 5.0$ ms (each frame requires $T_{HQ} + T_{mac} + T_{air} = 2 + 0.2 + 0.3 = 2.5$ ms). The overall latency is thus 9.0 ms.

Insight 2: When FA is enabled, retransmission (likely via software reschedule) is deferred longer than that via hardware retry. It also depends on how many packets have been waiting in the other software queues.

B. Empirical Study

We now use an empirical study to disclose how the high-goodput (HG) operations fail to minimize tail latency (we consider both 90^{th} and 95^{th} percentiles) in 802.11n. The operations include rate control (that decides packet loss and retransmissions), frame aggregation (that affects initial queueing delay and results in large latency due to software reschedule), and retransmission.

Experimental settings: We conduct experiments in a typical single family home environment (the left plot of Figure 4). The WiFi infrastructure mode is used. The AP is located at the spot$_{AP}$, whereas clients are placed at fine-grained spots from P0 to P15. Both AP and the clients use Atheros dual-band (2.4/5 GHz) 802.11n chipset, supporting up to three antennas and TS mode (the right picture in Figure 4). We use `iperf` to generate constant-rate UDP traffic. Unless explicitly specified, the default latency goal is the 90^{th} percentile tail, each traffic flow has 10 Mbps, and the packet size is 1470 bytes. For each test, we run 25 times and each run is long enough to collect 30K-packet traces. We present the result with the median PER.

Results: We consider the latency of the downlink flow from the AP to one observed client at P12. Similar findings are observed at other spots. Figure 5 plots the CDFs of packet delay for the client under two traffic patterns. In the left plot,

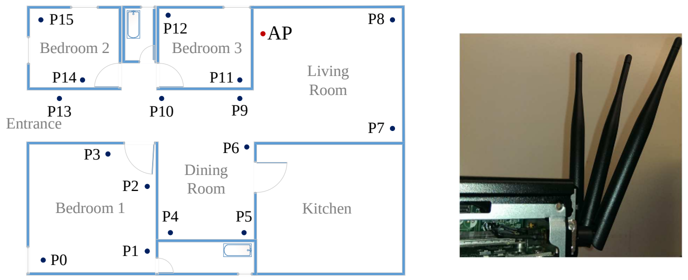


Figure 4: Left: home floorplan. Right: 802.11n platform.

there is only one observed downlink flow and no FA is used. In the right plot, there are six concurrent downlink flows to different clients, and by default, aggressive FA is adopted. We conduct exhaustive search to locate the rate settings yielding highest goodput (HG) and lowest latency (LL). In both cases, the HG and LL settings are 162DS[2] and 108DS. It is easy to see that HG performs worse given the 90^{th} (95^{th}) percentile latency goal.

Insight 3: The default HG operations in 802.11n may fail to achieve lowest latency.

We make three observations. First, retransmission (due to high PER) is critical to the tail latency (*e.g.*, the 90^{th}, 95^{th} percentile) (Insight 1). For the light traffic case with only one downlink flow (the left plot of Figure 5), HG performs better in terms of the medium/average latency. However, for the tail percentile, packets experience retransmissions at the HG rate (its PER is 17.9% and throughput is 85.4 Mbps). In contrast, LL has a much lower PER (1.7%) and negligible retransmissions, yielding shorter latency (*e.g.*, 0.23ms versus 0.54ms at the 90^{th} percentile). Under such light traffic (*i.e.*, each packet does not need to wait for others' transmission), initial queueing delay is negligible so that the time used for transmission and retransmission is more critical.

Insight 4: When initial queuing delay is negligible, the fastest rate setting **among those** *requiring the minimum retransmissions at the considered percentile, is preferred since retransmissions dominate the perceived latency.*

Second, FA aggravates the impact of retransmission. Under heavy traffic load (the right plot of Figure 5), the latency gap becomes wider. For the 90^{th} percentile tail latency, it turns into 2.7ms, up from 0.31 ms in case of no FA. This is because a software reschedule is employed when FA is used (Insight 2). The left plot in Figure 6 further explains why. It shows the number of software reschedules in the six-client setting. Using the HG rate, 12.2% packets encounters one reschedule, and 2.2% have at least two. It induces longer queueing delay. This queuing delay can be even larger, when there are more transmitters (*i.e.*, more clients have uplink traffic), which results in longer T_{HQ} for each transmission. The result of the scenario with one more uplink flow on top of the six-client case confirms this, as shown in Figure 1. It incurs extra 1.6 ms for the 90^{th} percentile tail latency.

Insight 5: Retransmission dispatching is required to reduce latency for software reschedule in case of retransmissions (which cannot be avoided in the high-PER scenario such as mobility, interference, etc.).

Third, the initial queuing delay increases with the number of transmissions preceding the packet of interest, especially with the same flow's ones. It is because the considered packet needs to wait for the drain of them, each of which experiences both T_{SQ} and T_{HQ}. We examine the effect of the preceding transmission number on the latency by varying the limit of aggregation size for the observed client using the HG setting. The number of preceding transmissions increases with the decreasing of the size limit. As shown in the right plot of Figure 6, the latency greatly increases with the decreasing

[2] 162DS denotes the 162 Mbps double-stream mode. Other MIMO modes include single-stream (SS), triple-stream (TS), and quadruple-stream (QS).

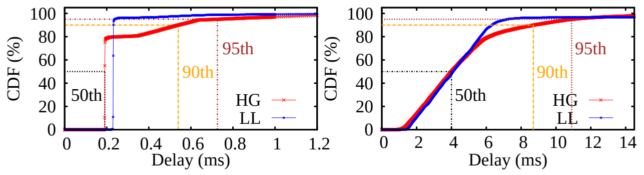


Figure 5: Latency in one-client (left) and six-client (right) cases.

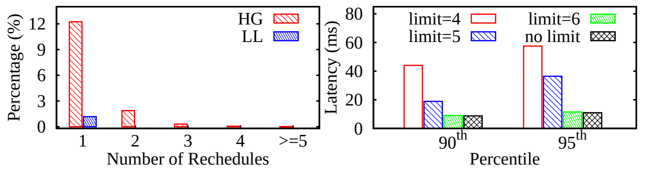


Figure 6: Left: Percentage of packets versus the reschedule number. Right: Latency versus the limit of aggregation size.

of the size limit in the six-client case, compared with the aggressive FA (No limit case). Therefore, to reduce the initial delay of the considered packet, the number of the transmissions preceding the packet should be made as small as possible.

Insight 6: Aggressive FA is preferred to reduce initial queuing delay.

In summary, the interplay of these three components in RA (*i.e.*, rate control, FA scheduling, retransmission dispatcher) exhibits inherent tradeoffs from the latency perspective. For rate control, the higher rate to use, the smaller airtime to transmit each packet, the possibly higher loss percentage to receive it. For FA scheduler, aggressive FA and the resulting larger aggregated frame size help to reduce the initial queueing delay but it also leads to the likelihood of an individual packet loss within a large frame. The resulting software reschedule by retransmission dispatcher generally incurs larger latency than hardware retry. We will address them in our design.

IV. LLRA Design

Our proposed RA has three components of rate controller, frame aggregation scheduler, and retransmission dispatcher, which work together to reduce tail latency. It runs at the AP. We assume that, an inter-client scheduler is responsible for allocating certain fair airtime to each client at the higher level. This can be readily achieved through temporal fair scheduling [11]. Note that ensuring low latency for uplink flows at one client is a subset of the problem at the AP, since the client only handles its own traffic, instead of the AP's multiple associated clients.

Our overall solution offers a unified operation of three components. Given the $\alpha-$percentile latency requirement (say, $\alpha = 90\%$), we split the considered N packets into two groups, one containing the packet with the latency at the $(\alpha \cdot N)^{th}$, and the other with the rest of packets. Each packet's latency consists of initial queuing delay (induced by packet arrival patterns) and service time (including both the airtime plus T_{mac} for transmission and the *hold* time due to retransmissions). Based on the six insights, we apply three rules:

- **Rule 1:** When queueing delay does not vary with rate settings (*e.g.*, only one transmission is needed to drain the queue), the service time thus makes difference. The fastest rate setting among those requiring the least number of retransmissions is preferred. Otherwise, we should consider the initial queueing delay and service time together for each rate setting.

- **Rule 2:** Aggressive aggregation is applied to the first group of packets, but not together with the second group.
- **Rule 3:** To offset the tail delay of the first group, prioritized reschedule is applied to the group's packets that require software reschedule.

In rate control, we seek to balance between the initial queueing delay and the service time. The general idea works as follows. In the presence of light traffic (e.g., only one aggregate frame in the software queue), we apply rate control that selects the highest rate setting with low PER, rather than the highest-goodput rate that incurs higher PER, to the aggregate frame. This is to reduce the retransmission-induced latency, which typically dominates the service time. Under heavy traffic (e.g., several aggregate frames are required to drain the queue), we apply rate control that balances the queueing delay and the service time, to reduce the initial queueing delay for the large data burst. In this case, the faster yet with larger PER rate may be better, because it drains packets faster.

Both low-latency FA scheduling and retransmission dispatching help to further reduce the queueing and service delays. FA scheduling applies aggressive aggregation to the first group's packets, to minimize the time of draining packets, thereby reducing the queueing delay. It separately applies aggressive aggregation to the second group, to minimize its effect on the queuing delay of subsequent packets. The retransmission dispatcher prioritizes the retransmitted frames from the first group when the queue is empty. It thus reduces the group's tail delay incurred by software reschedule without hurting others' initial queuing delay.

We next elaborate on each component.

A. LLRC: Low-Latency Rate Control

LLRC searches for the LL rate setting across different MIMO modes, and protects itself from the penalty of excessive probing. LLRC probes those rates which may reach the lowest latency at the α percentile, while applying several pruning rules. The search is triggered by both time-driven and event-driven approaches. With the former, LLRC periodically searches for the best one. With the latter, the event is that the current rate becomes worse. In the penalty protection, in addition to rate pruning, LLRC introduces a novel lightweight probing mechanism, which probes rates' PERs with few packets. We now present the LLRC search and pruning. Probing is described in Section IV-C. Note that we use the interference handling mechanisms similar to [19].

Pruning rules: We first derive three pruning rules for tail latency. These rules can speed up the search process by eliminating ineligible rates. They are specified by three corresponding theorems. Two prune lower and higher rate settings from the current best one R_{best} during search, respectively. The third rule is used for latency comparison between two rates. Given the packet latency as $D_{est} = T_q[0] + T_{srv}$. $T_q[0]$ is the initial queuing delay and depends on the number of preceding frames (N_{pre}) of the same flow. T_{srv} is the service time, and depends on the expected retransmission count (N_{rt}) and the

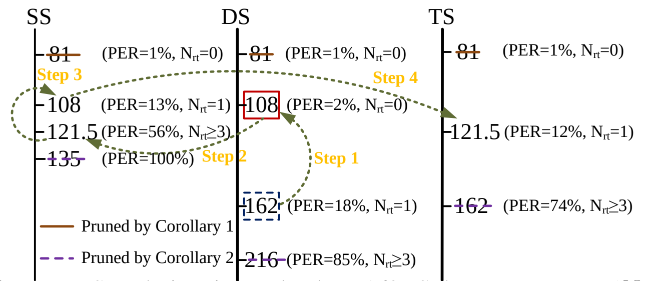


Figure 7: Search is triggered when 162DS becomes worse (N_{rt} changes from 0 to 1).

time consumed by each transmission (T_{tx}). The theorems are derived based on these parameters that vary with rate settings.

Theorem 1. *Given R_{best} with $N_{rt} = 0$, any low rate R (i.e., $R < R_{best}$) is pruned since it cannot perform better than R_{best} (i.e., $D_{est}(R) \geq D_{est}(R_{best})$).*

Proof: Since $R < R_{best}$ and R_{best} has low PER (due to $N_{rt} = 0$), R_{best} has smaller N_{pre} and T_{tx}. Moreover, it also has the smallest value of N_{rt}, 0, then $D_{est}(R) \geq D_{est}(R_{best})$ and R can be pruned. ■

Theorem 2. *Given a rate R which $R > R_{best}$ and $D_{est}(R) > D_{est}(R_{best})$, the higher rate R' (i.e., $R' > R$) in the same MIMO mode is pruned since it cannot perform better than $D_{est}(R_{best})$.*

Proof: Since $R > R_{best}$ and $D_{est}(R) > D_{est}(R_{best})$, R must have higher PER such that it has either of larger N_{pre} and larger N_{rt} or both. According to the steep increasing of PER between adjacent rates with non-negligible PER [19], the higher rate R' with much higher PER which increases both of N_{pre} and N_{rt} cannot perform better and thus can be pruned. Here, we do not consider T_{tx}, since N_{pre} and N_{rt} dominate the queuing delay and the service time, respectively. ■

Theorem 3. *Given two adjacent rates R_{low} and R_{high}. If they have identical N_{pre}, the one with smaller N_{rt} is better. If they also have the same N_{rt}, R_{high} is better. If they do not have identical N_{pre}, they need to be compared based on the estimated D_{est}.*

Proof: The same N_{pre} represents that R_{low} and R_{high} have the same initial queuing delay, so N_{rt} which dominates the service time leads the latency. Thus, the one with smaller N_{rt} has lower latency. When both N_{pre} and N_{rt} are the same, R_{high} transmits faster and thus has smaller T_{tx}, thereby achieving lower latency. ■

Search algorithm: LLRC differentiates MIMO modes (e.g., SS, DS, TS). Its search starts from the same MIMO mode of the current best rate setting (R_{best}). It proceeds along upward direction, and compares tail latency among adjacent rates by Theorem 3. LLRC replaces the current setting with a better one when one is found. This upward search continues until either the highest rate setting is reached or all higher rates are pruned by Theorem 2. It then queries downward from the original best rate setting if the number of retransmissions (denoted as N_{rt}) is one or more; otherwise, all lower rates are pruned ($N_{rt} = 0$) based on Theorem 1. It continues until either the lowest rate setting or the highest one with $N_{rt} = 0$ is reached. After the current MIMO mode, LLRC prunes the rate settings at other MIMO modes by using the highest rate with $N_{rt} = 0$. By Theorem 1, all rates lower than this one can be pruned. In each mode, the search starts from the rate which is next higher than the current best one. It searches upward and then downward, with the same stop conditions.

We illustrate the algorithm via the example of Figure 7. It is triggered by the event that 162DS becomes worse. Assume N_{rt} changes from 0 to 1 when the client moves from P10 to P12 (Figure 4). We consider the 90^{th} percentile tail latency. Assume that $N_{pre} = 0$. The initial queuing delays thus are the same among different settings. It first searches downward in the same MIMO mode, so it queries 108DS (Step 1) and changes the best rate to it, because its $N_{rt} = 0$ incurs lower latency than 162DS with $N_{rt} = 1$. With identical initial queuing delay, N_{rt} dominates the overall latency. The rates higher than 162DS can be pruned by Theorem 2. Moreover, by Theorem 1, all rates lower than 108DS can also be pruned among all MIMO modes. The search of this mode thus stops. LLRC then searches the next higher rate, 121.5SS, in the SS mode (Step 2). Since it is worse than 108DS due to $N_{rt} \geq 3$, the rates higher than 121.5SS are pruned. It stops at 108SS, which is worse due to $N_{rt} = 1$, (Step 3) in this mode. It further probes the next higher rate, 121.5TS, in the TS mode (Step 4), and prunes higher rate settings. The search completes with the best rate, 108DS.

B. Aggregation Scheduling and Retransmission Dispatching

We divide packets into two groups based on the retransmission count occurring at the packet of the α percentile. It dominates the latency metric, given the same queuing delay incurred by the current best rate. We collect N_{rt} and the latency of N packets in history, and obtain the expected retransmission count N_{rt} of the packet (say, β) at the α percentile. The packets with retransmission count no larger than β belong to the first group, which contains the packet with latency at the $(\alpha \cdot N)^{th}$. The remaining packets belong to the second group.

Aggregation scheduling packs the first group's packets as many as possible for transmission. Once no more packets in the first group are queued, aggressive FA is applied to the second group's packets. If any packet in the first group encounters software reschedule, it is given the highest priority and scheduled to the hardware queue right after its failure. It thus reduces the tail delay of the first group without increasing the initial queuing delay of other packets in this group.

Take Case IV of Section III-A as an example. In Figure 3, we consider two cases: $\beta = 0$ and $\beta = 1$. Assume that the maximum aggregation size of the current rate is larger than 3. In both cases, aggressive aggregation is applied into the aggregation formation of packets 1, 2, and 3, since all have no retransmission count and belong to the first group. However, different actions are taken for the retransmission of Packet 3 in two cases. In the former case, it belongs to the second group due to $\beta = 0$. It is thus not prioritized for retransmission and takes 9.0 ms. In the latter case, it belongs to the first group due to $\beta = 1$. Its retransmission is then prioritized over other traffic. Consequently, it does not wait for other frames in the software queues, thereby taking only 4 ms.

N_{rt}	1	2	3
$\alpha = 90^{th}$	10.0%	31.6%	46.4%
$\alpha = 95^{th}$	5.0%	22.3%	36.8%

Table II: PER threshold versus retransmission count.

History Probing	121.5SS	162DS	121.5TS
1	1	1	1
2	1	1	0
3	0	0	0
4	1	0	1
5	1	1	1

Table III: Example of five probing transactions for three rates when 108DS is being used. 0 and 1 denote applicable and inapplicable, respectively.

C. Novel Light-weight probing

We now introduce a novel light-weight probing mechanism; the goal is to probe PER at a given rate. For example, the 95^{th} percentile requires PER to have accuracy no larger than 5% (*i.e.*, at least 20 packets). There are two general approaches to PER estimation: probing-based [19] and SNR-based [13]. Most current real platforms do not support the SNR scheme. We thus take the probing-based approach, in order to implement and test it on real systems.

It consists of three components: slow-start pruning, association rule-based pruning, success inheritance. The first two eliminate probing packets for inapplicable rates, especially those high-loss ones. The last one reduces probing of applicable rates. A rate may inherit successful probings from the higher rate, thereby reducing its own probing. If other flows without latency requirements coexist with the latency-sensitive one, their packets are used for probing. The latency-sensitive flows accordingly have less probing overhead.

Slow-start pruning: It detects an inapplicable rate with few packets. To probe a rate, it starts from the frame with one-packet size, and then exponentially increases the frame size (*i.e.*, two-packet, four-packet, and so on). It stops when either the collected results are sufficient or the packet loss count has verified the inapplicability of the rate. A rate is inapplicable when its PER exceeds certain threshold associated with the retransmission count, such that it can perform no better than the current best rate, according to Table II (to be derived later).

Association rule-based pruning: It uses correlations among rates to infer a rate's inapplicability from other rates under similar channel conditions. The underlying premise is that, each rate setting behaves similarly among time-varying samples under the same channel condition, so rate settings can be correlated with each other in their performance. We apply association-rule learning to discover correlations between rate settings. Such correlations are learned from the historical probing statistics. We further obtain the confidence level of each correlation, which indicates its reliability. If the confidence level of one correlation μ is larger than a specified value (say, $\mu \geq 0.9$), the correlation is dependable. We only apply the dependable correlations in our work. Finally, once the dependable correlation of each rate pair is learned, one rate's result can be used to infer the other's. That is, one rate's failure infers that the other rate also fails.

Table III shows an example of five probing transactions for three candidate rates (108DS is the used, current rate). The inapplicable rates are marked as 1; otherwise, they are set to 0. The correlation "if 162DS fails (*i.e.*, inapplicable), then 121.5SS fails" has the confidence of 1.0 ($\frac{fail(162DS \cup 121.5SS)}{fail(162DS)} = \frac{3}{3}$), and can be used. However, the one "if 162DS fails, then 121.5TS fails" has the confidence of 0.7 ($\frac{2}{3}$), and is not applicable. Consequently, if the probing of 162DS fails, we do not probe 121.5SS.

D. Parameter Estimation

We further estimate various parameters used in LLRA. The initial queuing delay includes both software and hardware queueing delays. Assume that the α percentile's packet is always in the last frame, which empties the queue, and aggressive FA is applied. It can be estimated as follows, based on the average number of simultaneous packets in the queue and the rate's maximum aggregation size:

$$T_q[0] = (N_{pre} + 1) \times T_{sw,interval} + T_{HQ}, \tag{2}$$

where $T_{sw,interval}$ represents how often a frame at the head of the software queue (SQ) is scheduled to the hardware queue (HQ). The software queuing delay is estimated based on the number of preceding frames (N_{pre}) plus 1 (*i.e.*, itself) and the scheduling interval ($T_{sw,interval}$). T_{HQ} denotes the hardware queuing delay.

Service time depends on the retransmission count N_{rt} and type. Given the expected retransmission count at the α percentile with the given rate's PER. It can be estimated in both non-FA ($T_{srv,nofa}$) and FA ($T_{srv,fa}$) cases:

$$\begin{aligned} T_{srv,nofa} &= T_{tx} + N_{rt} \times (T_{hw,interval} + T_{tx}), & (3)\\ T_{srv,fa} &= T_{tx} + N_{rt} \times (T_{HQ} + T_{tx}), & (4)\end{aligned}$$

where T_{tx} is the sum of airtime and MAC overhead for each transmission, and $T_{hw,interval}$ represents how often a frame at the head of HQ is scheduled for transmission. $T_{hw,interval}$ is usually smaller than T_{HQ}, since the size of HQ may be larger than one frame. In $T_{srv,fa}$, T_{HQ} is used for the retransmission of software reschedule. Low-latency retransmission dispatching places the rescheduled packets to the tail of HQ, instead of SQ, right after failure. For simplicity, T_{tx} is treated the same between different transmissions, since its variation is negligible compared with the number of retransmissions, which dominates the service time.

Given the PER at a given rate setting, we estimate the retransmission count (N_{rt}) for the packet at the α percentile. Assume packet error events are independent and have the same probability p. The probability that a successful transmission is preceded by i failures, is given by $p^i(1-p)$. This probability also represents the percentage of the successful frames after $i+1$ transmissions. If the percentage is larger than α, we compute N_{rt} as the smallest integer satisfying $\sum_{i=0}^{N_{rt}} p^i(1-p) \geq \alpha$. As a result, we can calculate the PER threshold for the retransmission count shown in Table II.

Moving average estimation is applied for the following parameters: N_{pre}, T_{tx}, T_{HQ}, $T_{hw,interval}$, and $T_{sw,interval}$.

Given rate setting R, N_{pre} is estimated as $\left\lceil \frac{N_{sp} \times \rho}{maxFA(R)} \right\rceil - 1$, where N_{sp} is the number of simultaneous first-attempt packets appearing together at the queue, ρ represents the ratio of the first-attempt packets to the total number of packets with no more than three attempts, and $maxFA(R)$ is the maximum aggregation frame size of R. Note that the number of packets with more than three attempts is negligible. To estimate N_{sp}, we start to count packets when they come to the empty queue, but stop counting when no more first-attempt packet is in the queue after a frame is scheduled. T_{tx} is given by $\frac{avgFA(R) \times framesize}{tpt(R)} + T_{mac}$, where $avgFA(R)$ is the average aggregation frame size of R, $framesize$ is the average frame size, and $tpt(R)$ is the loss-free throughput of R. $avgFA(R)$ can be calculated as $\frac{N_{sp} \times \rho}{N_{pre}+1}$.

$T_{sw,interval}$ takes into account both the traffic load at the AP and that from other clients on the same channel. It is estimated based on the scheduling interval of two adjacent frames in the SQ. $T_{hw,interval}$ is estimated in two cases. If HQ does not have any frame ahead of the current transmission, we compute $T_{hw,interval}$ as the duration from the time the frame is placed into HQ to the time its ACK is received, while subtracting transmission airtime (T_{air}) and the MAC overhead (T_{mac}). If HQ has frames before the current transmission, $T_{hw,interval}$ is calculated as the duration between two consecutive ACKs for first-attempt transmissions, while also excluding T_{air} and T_{mac} of the second frame. Finally, for the hardware queueing delay T_{HQ}, it is the duration from the time the frame is just placed into HQ to the instant its ACK is received, which excluding T_{air} and T_{mac}.

E. Other Issues

There are some additional issues in LLRA.

Coexistence of LLRA and other RA goals at different clients: LLRA may coexist with the clients pursuing other goals (*e.g.*, highest goodput and energy efficiency) under the same AP. The LLRA clients may slow down from their high-goodput rate settings and grab more airtime, thereby possibly affecting the performance of other clients (*e.g.*, lower goodput or less energy-saving). Nevertheless, the impact can be alleviated or eliminated through an inter-client scheduler. The airtime allocated to each client is constrained by the scheduling policy (*e.g.*, fair airtime allocation via temporal fair scheduling [11]). Thus, each client is only able to use up its assigned airtime so that it does not affect the others no matter which goal it pursues. When LLRA clients do not have enough airtime for the slowdown of LL rate settings, they switch to the highest-goodput settings upon their backlogged queue. Note that the scheduling mechanisms and policies of airtime allocation are independent of this work.

Impact of LLRA on non-latency-sensitive flows at the same client: LLRA may hurt the throughput of the non-latency-sensitive flows, when LLRA is simply applied to all the flows at one client. It can be addressed by considering latency-sensitive flows and the others separately. For example, a hybrid RA algorithm pursues the lowest latency for the former flows, while retaining the highest goodput for the latter ones. We leave it to the future work.

V. Implementation

We implement our design in Atheros 802.11n WiFi driver (*i.e.*, `ath9k` [1]). At the initialization phase, the client MAC is notified of the low-latency requirement for a given flow based on its five-tuple flow ID. The wireless MAC then notifies the AP of this flow requirement via an action frame, a type of management frame. To measure the latency of a packet, we rely on the structure of its associated socket buffer to hold the start timestamp. The timestamp is logged once the packet is passed to the MAC from the upper layer. It is then traced via the sequence number in the MAC header, when it moves between hardware and software queues, transmission, and retransmission. When its ACK is received, the difference between the logged start timestamp and the current one is the overall latency. We execute our design mainly in two modules of the WiFi driver: rate control and transmission modules. The former module implements rate control, whereas the latter takes charge of the FA scheduler and the retransmission dispatcher. Note that our implementation not only uses standard-compliant messages, but also follows the general WiFi driver framework. It can be readily ported to other WiFi drivers.

VI. Performance Evaluation

We evaluate LLRA through extensive experiments in a commodity 802.11n network. We compare LLRA with two 802.11n RA algorithms – default Atheros RA (ARA) [23] and MiRA [19]. We examine the latency for a downlink flow in both single-client (non-interference, interference, and static and mobile scenarios) and multi-client cases. Experimental settings are described in Section III-B. Findings are similar. We only present some representative results due to space limit. Our evaluation shows that LLRA consistently outperforms ARA and MiRA in all tested scenarios. For example, it reduces the 90^{th} percentile latency by 1–19 ms in interference and multi-client cases (about 22–68% reduction).

A. Single-client Case

We first consider the non-interference cases over the 5GHz band at various spots or with different latency goals. We further assess it in the interference and mobility cases.

Client Placements: We study LLRA under various wireless channels by placing the client at different spots. Figure 8 shows the 90^{th} percentile tail latency and the achieved goodput at various spots. LLRA consistently outperforms ARA and MiRA by reducing latency by 30.7%–75.2% and 22.9%–63.4%, respectively. Moreover, it only sacrifices 0.5%-13.3% goodput. It is because LLRA chooses LL rates at low probing costs. Figure 9 plots the rate distribution at P3 and P6. The rates chosen by LLRA are a little slower than the HG rates but have lower PERs. Take P3 as an example. LLRA mainly stays at the LL rate, 121.5TS, which has comparatively low PER (1.9%), while also oscillating between two adjacent rates, 162DS and 108DS. In contrast, ARA and MiRA tend to choose the HG rate (162DS), which results in higher PERs ($\approx$ 12%).

Latency Goals: We vary latency goals to examine how LLRA performs with different tail latency requirements. We use P14 as an example to demonstrate its impact. The results are similar to other spots. Figure 10(a) shows the latency gain for three percentiles, 80^{th}, 90^{th}, and 95^{th}. We have two observation. First, LLRA outperforms both ARA and MiRA for all three goals. Second, the gain becomes smaller at the lower percentile. The latency reduction decreases from 63.4% to 31.9% (ARA) and from 52.5% to 8.0% (MiRA), when the percentile changes from 95^{th} to 80^{th}. It is because the chosen LL rate is very close to the HG one, as the percentile decreases.

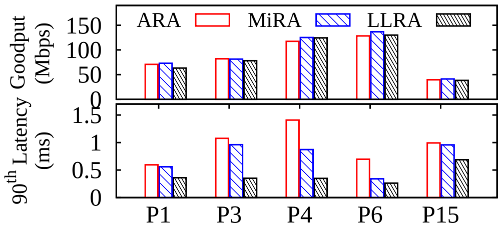


Figure 8: 90^{th} percentile latency and goodput vary with spots.

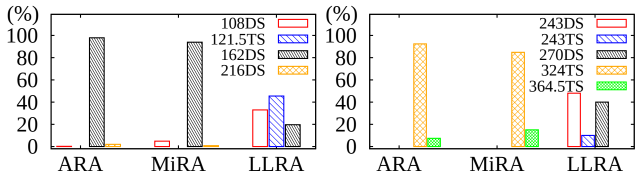


Figure 9: Rate distributions at P3 (Left) and P6 (Right).

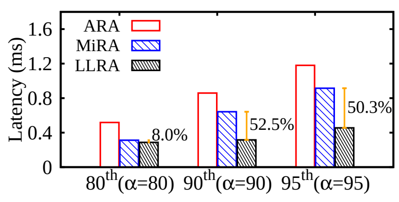
Interference: We evaluate LLRA with rich interference using Channel 1 (20MHz) at the crowded 2.4GHz. It is observed that there are more than 10 APs from neighbors. Figure 10(b) shows the 90^{th} latency at P3 and P9. LLRA is still more effective in lowering latency. The reduction over ARA is 19 ms (67.7%) and 3 ms (50.6%), while the one over MiRA is 4 ms (32.3%) and 1 ms (21.8%). The most noticeable thing is that actual latency is much larger in this case, *e.g.*, 28 ms for ARA, up from 1 ms in the non-interference case. The reason for much larger delay is that (1) interference brings longer queuing and contention time, and (2) RA algorithms tend to slow down due to interference-induced loss. Even so, LLRA performs robust against co-channel interference. It implies its effectiveness in competing scenarios.

Mobility: We move the client from P6 to P1 along the wall and then back to P6, at the constant pedestrian speed of 1 m/s. The client is loaded with 5 Mbps source. Figure 10(c) shows that LLRA outperforms ARA and MiRA by 85.3-90.8% and 42.6-66.2%, respectively. It implies that LLRA is able to rapidly locate the LL rates during mobility.

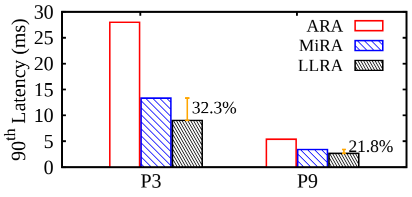
B. Multi-client Case

We further evaluate LLRA in the multi-client case. We consider two sub-cases: the observed client is placed in a far spot (P0) or a near spot (P13). Another five clients are deployed at P4, P6, P7, P9, and P10. Each client is loaded with a downlink traffic flow. We use up to 4 clients (P0) and 6 clients (P13). Figures 11(a) and 11(b) show the results for the 90^{th} and 95^{th} percentile latency. Even compared with MiRA (better than ARA), LLRA reduces latency by 30.5%–52.5% (2–5 ms). LLRA prefers to stay at the LL rate without incurring retransmissions at both 90^{th} and 95^{th} percentiles. However, ARA and MiRA stay at the HG rates and aggressively probe higher rates with high losses. Their high PERs (15.1-29.7%) result in one or two retransmissions, incurring larger delay than the single-client case.

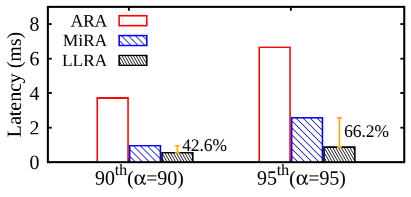
In order to examine how retransmission dispatcher comes into play, we test LLRA with high loss in the multi-client case. We induce more possible collisions by adding one or



(a) Various Percentiles at P14

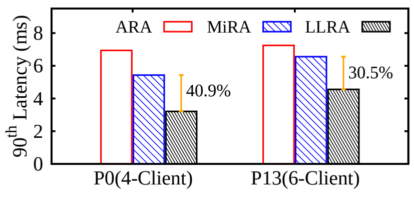


(b) Interference Case

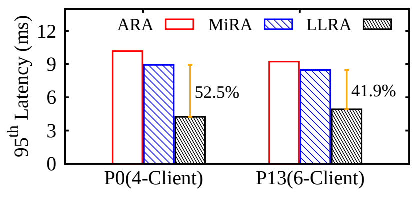


(c) Mobility Case

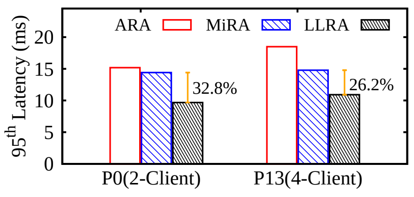
Figure 10: The tail latency varies with different scenarios in the single-client case.



(a) $\alpha = 90$ at observed client



(b) $\alpha = 95$ at observed client



(c) $\alpha = 95$ with uplink traffic

Figure 11: Multi-client case. (a)(b): a downlink flow per client. (c): downlink flows plus one (P0) or two (P13) uplink flows.

two uplink flows. The uplink traffic also reduces the capacity for downlink flows; we then support 2-clients and 4-clients in the P0 and P13 cases. Figure 11(c) gives the 95^{th} percentile latency for the observed client. LLRA continues to outperform them by saving 4–8 ms (26.2%-41.0%). It is mainly attributed to its retransmission dispatcher. Even though the rate controller chooses the rate with lower PER, the packet at the 95^{th} percentile for LLRA still experiences one retransmission via software reschedule. Retransmission dispatcher then helps to prioritizes retransmissions to have smaller queueing delay.

VII. Conclusion

Supporting latency-sensitive applications is important for 802.11n/ac home networks. As 802.11n/ac is increasing its link speed, the common-sense perception "*higher speed is enough to ensure low latency for such applications*" is plain wrong. The fundamental problem lies in the long-tailed latency distribution at the link layer. Current link rate adaptation for 802.11n devices works well for throughput, but cannot meet the millisecond latency demand. In this work, we design LLRA, which dynamically balances between various settings ranging from *"higher goodput yet higher loss"* to *"slightly lower goodput but much lower loss"*. Through extensive evaluation, LLRA outperforms traditional high-goodput RA to reduce the tail latency.

References

[1] ath9k. http://wireless.kernel.org/en/users/Drivers/ath9k.

[2] Chromecast. http://www.google.com/chrome/devices/chromecast/.

[3] Chromecast Gaming. http://www.technologytell.com/gaming/123089/streaming-games-through-googles-chromecast-is-now-a-big-possibility/.

[4] Razer Gaming Mouse. http://www.razerzone.com/gaming-mice.

[5] IEEE 802.11e Standard: MAC and PHY Specifications Amendment 8: MAC Quality of Service Enhancements, 2005.

[6] IEEE 802.11n Standard: MAC and PHY Specifications Amendment 5: Enhancements for Higher Throughput, 2009.

[7] IEEE 802.11ac Standard: MAC and PHY Specifications Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz, Jan 2011.

[8] S. Analytics. Global broadband and wlan (wi-fi) networked households forecast 2009-2018, Oct 2014. http://www.strategyanalytics.com/default.aspx?mod=reportabstractviewer&a0=10232.

[9] C. Chafe et al. Effect of Time Delay on Ensemble Accuracy. In *ISMA'04*.

[10] A. Chan et al. Video-Aware Rate Adaptation for MIMO WLANs. In *ICNP'11*.

[11] H. M. Chaskar et al. Fair Scheduling with Tunable Latency: A Round-robin Approach. *IEEE Transactions on Networking*, 11(4), 2003.

[12] Z. Feng et al. Trading Off Distortion for Delay for Video Transmissions in Wireless Networks. In *INFOCOM'13*.

[13] D. Halperin et al. Predictable 802.11 Packet Delivery from Wireless Channel Measurements. In *SIGCOMM'10*.

[14] K.-Y. Jang et al. Snooze: Energy Management in 802.11n WLANs. In *CoNEXT'11*.

[15] C.-Y. Li et al. Energy-based rate adaptation for 802.11n. In *MOBICOM'12*.

[16] J. Manweiler et al. Switchboard: A matchmaking system for multiplayer mobile games. In *MobiSys*, 2011.

[17] V. Namboodiri et al. Alert: An Adaptive Low-Latency Event-Driven MAC Protocol for Wireless Sensor Networks. In *NSDI'12*.

[18] W. Pasman et al. Latency layered rendering for mobile augmented reality. In *ISIT'00*.

[19] I. Pefkianakis et al. MIMO Rate Adaptation in 802.11n Wireless Networks. In *MOBICOM'10*.

[20] M. V. D. Schaar et al. Optimized Scalable Video Streaming over IEEE 802.11a/e HCCA Wireless Networks under Delay Constraints. *IEEE TMC*, 2006.

[21] A. Vulimiri et al. Low Latency via Redundancy. *CoRR*, 2013.

[22] K. Winstein et al. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *NSDI'13*.

[23] M. Wong et al. Wireless LAN using RSSI and BER Parameters for Transmission Rate Adaptation, 2008. US patent, 7,369,510.

[24] X. Xu et al. A Delay-Efficient Algorithm for Data Aggregation in Multihop Wireless Sensor Networks. *IEEE TPDS*, 2010.

[25] W. Zhou et al. ASAP: A Low-latency Transport Layer. In *CoNEXT'11*.