Frame Retransmissions Considered Harmful: Improving Spectrum Efficiency Using Micro-ACKs

Jiansong Zhang *
Microsoft Research Asia and HKUST
Beijing, China
jiazhang@microsoft.com

Haichen Shen *
Microsoft Research Asia and Tsinghua University
Beijing, China
v-hashen@microsoft.com

Kun Tan
Microsoft Research Asia
Beijing, China
kuntan@microsoft.com

Ranveer Chandra
Microsoft Research
Redmond, WA, USA
ranveer@microsoft.com

Yongguang Zhang
Microsoft Research Asia
Beijing, China
ygz@microsoft.com

Qian Zhang
HKUST
Hong Kong, China
qianzh@cse.ust.hk

ABSTRACT

Retransmissions reduce the efficiency of data communication in wireless networks because of: (i) per-retransmission packet headers, (ii) contention overhead on every retransmission, and (iii) redundant bits in every retransmission. In fact, every retransmission nearly doubles the time to successfully deliver the packet. To improve spectrum efficiency in a lossy environment, we propose a new in-frame retransmission scheme using μACKs. Instead of waiting for the entire transmission to end before sending the ACK, the receiver sends smaller μACKs for every few symbols, on a separate narrow feedback channel. Based on these μACKs, the sender only retransmits the lost symbols after the last data symbol in the frame, thereby adaptively changing the frame size to ensure it is successfully delivered. We have implemented μACK on the Sora platform. Experiments with our prototype validate the feasibility of symbol-level μACK. By significantly reducing the retransmistion overhead, the sender is able to aggressively use higher data rate for a lossy link. Both improve the overall network efficiency. Our experimental results from a controlled environment and an 9-node software radio testbed show that μACK can have up to 140% throughput gain over 802.11g and up to 60% gain over the best known retransmission scheme.

Categories and Subject Descriptors

C.2.1 [**COMPUTER-COMMUNICATION NETWORKS**]: Network Architecture and Design—*Wireless communication*

General Terms

Algorithms, Design, Experimentation, Performance

*This work is done when Jiansong Zhang is a researcher and Haichen Shen is a research intern in Microsoft Research Asia.

MobiCom'12, August 22–26, 2012, Istanbul, Turkey.

Keywords

Retransmission, Error Recovery, Feedback, μACK, WLAN

1. INTRODUCTION

Packets transmissions over a wireless medium are lossy. Losses typically occur because of interference, fading or noise, which in turn causes poor signal to interference and noise ratio (SINR) at the receiver. A low SINR reduces the probability of successfully decoding all bits in the packet.

In existing data networks, such as Wi-Fi or cellular, the onus of recovering a lost packet is with the link layer of the OSI stack. The physical layer (PHY) may include some redundancy, *i.e.*, channel coding, in a frame to correct erroneous bits. But when the errors cannot be recovered, the frame will be retransmitted by the MAC layer. The sender usually relies on an acknowledgment (ACK) packet from the receiver to detect transmission failures. For example, in IEEE 802.11, if the sender does not receive the ACK packet within a fixed timeout period, it retransmits the entire frame.

This frame retransmission is costly. First, each retransmitted frame requires both PHY and MAC headers, which can consume up to 100 bytes. While MAC header can be transmitted using high modulation, the PHY header is sent at the lowest rate and usually includes a sequence of training symbols (preamble) for synchronization and channel estimation. In 802.11g, this header overhead can add up to 52 μs. Second, the sender needs to re-contend for the medium to retransmit the packet. Depending on the number of additional contending nodes, the sender has to perform backoff, possibly several times, before the frame can be successfully delivered to the receiver. For each unsuccessful retransmission, the sender needs to wait for an ACK timeout before it can detect the loss. All reduce the link efficiency in a noisy wireless channel. Finally, retransmitting the entire frame may unnecessarily send redundant bits that may have already been received correctly by the receiver. In previous work, Jamieson, *et. al.*, tried to address the last issue by selectively requesting only erroneous bits in retransmission [12]. But the first two issues remain unexplored.

In this paper, we propose a system that does away with frame-level retransmissions. After a sender gets access to a medium, we dynamically adjust the length of the packet (up to a maximum limit) to ensure that it is reliably delivered. In a zero-loss network the packet length is unchanged; while in a lossy network we pad the

frame, on the fly, with the bits that are received erroneously at the receiver. Therefore, the receiver can recover the errors inside a frame, instead of waiting for another frame retransmission. Since the retransmitted symbols do not have additional PHY/MAC headers or contention overhead, they significantly reduce the cost of error recovery. As we show in Section 6, our system can be 140% more efficient than 802.11, and up to 60% better than the existing best known retransmission scheme [12].

We achieve this *in-frame error recovery* by introducing a narrowband feedback channel, similar to the control channels proposed in RI-BTMA [21] and other schemes [16]. However, instead of transmitting a simple tone (or signature signal), in our system, the receiver uses the control channel to send modulated acknowledgments to the sender for received symbols. We call these tightly synchronized symbol-level acknowledgments as *micro-ACKs* (μACKs).

Implementing the μACK system imposes several challenges. First, the forward and feedback radios should be tightly synchronized. The receiver needs to dynamically generate μACKs based on the decoding results of the data symbols. The sender needs to re-encode erroneous bit based on μACK feedbacks. This needs to occur in real-time in the order of a few symbol durations (tens of μs). Second, the receiver needs to reliably determine the data symbols that are correct or in error. Previous work [12] uses PHY hints to identify erroneous symbols. Although useful in several scenarios, this scheme is not reliable when the link is operating at the modulation's threshold SNR, *i.e.*, when retransmissions are more likely to occur.

This paper presents the design, implementation, and evaluation of μACK. μACK uses a multi-radio architecture with multiple RF front-ends tightly integrated onto one control board. We further exploit a new side-channel inside 802.11 OFDM PHY to transmit a CRC-checksum along with a group of data symbols to facilitate error detection without adding additional overhead. Finally, we design and evaluate the PHY schemes for both, the side control channel and the μACK feedbacks. We show the μACK PHY design is simple, yet reliable for their purpose in our system.

We implement μACK using a high speed software radio platform [18]. Experiments with our prototype validate that symbol-level μACK is practically feasible. We also show that with μACK, the retransmission overhead can be significantly reduced, thereby improving spectrum efficiency. Furthermore, we believe that the idea of μACK has wider applicability, beyond error recovery. For example, using μACK, a sender can detect collisions before the entire transmission is complete, and therefore it can abort earlier to save channel time, similar to [16]. Also, the μACK feedback channel can be used as an extended busy-tone [6], and therefore can mitigate hidden and expose terminal problems.

The rest of paper is organized as follows. Section 2 motivates our work with an analysis of the retransmission overhead. Section 3 presents the detailed design of μACK. We further analyze μACK in Section 4. After describing our implementation of μACK using a high-speed SDR platform in Section 5, we evaluate the performance of μACK in Section 6. Section 7 discusses related work and Section 8 concludes.

2. OVERHEAD OF RETRANSMISSIONS

Although retransmissions help recover a packet, they add significant redundancy and overhead, thereby reducing spectrum efficiency. In this section we present a simple model to quantify this overhead, which motivates the need for the μACK mechanism.

Model: We consider the impact of retransmissions on IEEE 802.11g networks, although the results can similarly be extrapolated for 802.11a/n/b networks. In 802.11g, the timing parameters are, $t_{slot} = 9\mu s, t_{SIFS} = 10\mu s$, and $t_{DIFS} = 2t_{slot} + t_{SIFS} = 28\mu s$. The contention overhead, determined empirically from Atheros cards is $t_{CW} = t_{slot} \cdot \frac{CW_{min}}{2} = 8t_{slot}$ per packet. Each OFDM data symbol is 4 μs.

Using the model presented in [9], at modulation rate R, $4 \cdot R$ data bits are encoded in a symbol. The frame is broken down and encoded in symbols of duration $t_{symb} = 4\mu s$, where each frame is preceded by a 20 μspreamble, and a 6 μs signal extension. Using all these values, the time to transmit a packet of size s_{data} bits at data rate R Mbps, without any retransmissions is:

$$\begin{aligned} T_{data}(R) &= t_{CW} + t_{DIFS} + t_{DATA} + t_{SIFS} + t_{ACK} \\ &= 72\mu s + 28\mu s \\ &\quad + (20\mu s + t_{symb}\lceil s_{data}/(4R)\rceil + 6\mu s) + 10\mu s \\ &\quad + (20\mu s + t_{symb}\lceil s_{ack}/(4R_{ack})\rceil + 6\mu s) \\ &= 162 + 4 \cdot (\lceil s_{data}/(4R)\rceil + \lceil s_{ack}/(4R_{ack})\rceil)\mu s \end{aligned}$$

When there are r retransmissions, the time to transmit a packet is:

$$\begin{aligned} T_{ReTx} &= \sum_{i=0}^{r} T_{data}(R_i) \\ &\geq 162(r+1) + 4(r+1) \cdot (\lceil s_{data}/(4R)\rceil \\ &\quad + \lceil s_{ack}/(4R_{ack})\rceil)\mu s \end{aligned}$$

As we see, retransmissions delay the total time to transmit the frame due to the following:

- **Redundant bits ($T_{Redundancy}$):** These are bits that are correctly received by the receiver, yet are part of the retransmission. Depending on the number of bits in error, this adds anywhere from 0 (all bits are lost) to $(4r \cdot (\lceil (s_{data} - 1)/(4R)\rceil)$ when only 1 bit is decoded error.

- **Contention ($T_{Contention}$):** A retransmission has to contend for the medium with all other nodes in the medium, just like a fresh transmission. This adds the DIFS and contention overhead, for a total of: $(100 \cdot (r-1))$ μs.

- **Header overhead (T_{Header}):** Since the retransmission is just like a new frame, it has to include all training symbols, the PHY and MAC header, as well as an ACK frame. This adds another $((r-1) \cdot (62 + \lceil s_{ack}/(4R_{ack})\rceil))$ μs.

- **Lower data rate ($T_{DataRate}$):** Bits in the retransmission are usually sent using a lower data rate. In most common implementations, the retransmission data rate is one rate lower than the original data rate [2]. For example, when the original packet is sent at 18 Mbps, and the retransmission at 12 Mbps, the extra overhead is: $(r-1) * (T_{data}(12) - T_{data}(18))$.

To get a quantitative feel of these numbers, we present the overhead for a single retransmission in Table 2. We change the packet size and data rate, and note that the retransmission consumes more time than the original transmission because retransmissions are sent at a lower data rate. The contention overhead is assumed to be 100 μs. It will be much larger in congested environments. Also, $T_{Redundancy}$ is assumed to be the worst case, *i.e.*, all but one bit is erroneously decoded. As expected, the contention and header overheads are a much larger fraction of the retransmission overhead when the packets are small in size, or the data rates are higher. This fraction is likely to dominate in IEEE 802.11n MIMO networks with much higher data rates. To summarize, even one retransmission more than doubles the delivery time of a packet, and also reduces the spectrum efficiency because of the factors highlighted above.

(Pkt size, Data Rate)	$T_{data}(\mu s)$	$T_{ReTx} - T_{data}(\mu s)$	$\frac{T_{Redundancy}}{T_{ReTx}}$	$\frac{T_{Contention}}{T_{ReTx}}$	$\frac{T_{Headers}}{T_{ReTx}}$	$\frac{T_{DataRate}}{T_{ReTx}}$
(1500, 54)	389	417	0.53	0.24	0.16	0.07
(1500, 9)	1500	2166	0.61	0.05	0.03	0.31
(500, 54)	241	250	0.30	0.40	0.26	0.04
(500, 9)	611	833	0.53	0.12	0.09	0.26

Table 1: The overhead introduced by one retransmission on changing packet size (in bytes) and the data rate (in Mbps) of the original transmission, and the fraction of overhead introduced by each of the four factors. Contention and packet headers dominate for smaller packets, while redundancy is the largest overhead otherwise.

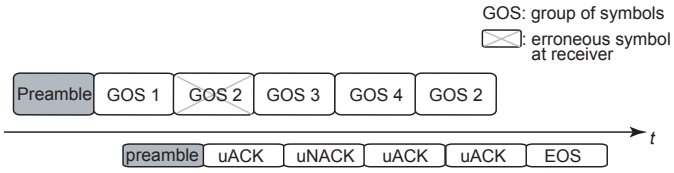


Figure 1: Illustration of a frame transmission in μACK system.

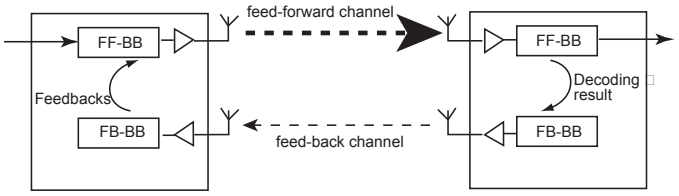


Figure 2: Architecture of μACK. Two tightly synchronized radios are deployed at both sender and receiver. The receiver sends real-time feedback symbols to the sender over a narrow-band channel.

3. μACK DESIGN

In this section, we present our system, that eliminates packet retransmission and thereby mitigates the overhead in recovering errors. The receiver decodes each incoming data symbol and dynamically determines if the symbol is correct or in error. For a group of successfully decoded symbols, the receiver will send an acknowledgment (ACK) to the sender; otherwise, if erroneous symbols are detected, an negative acknowledgment (NACK) is sent. These ACK/NACK are transmitted back to the sender using a separate narrow-band feedback channel, which is tightly synchronized to the feed-forward data communication channel. Since the ACK/-NACKs are at symbol level, we call them micro-ACKs/NACKs (or μACKs/ μNACKs). The sender monitors the feedback channel and marks the symbols that get NACKs. Then, when all data symbols have been sent, the sender re-encodes these symbols and appends them after the last data symbol. The sender continually re-encodes the lost symbols until it receives an acknowledgment of the entire frame from the receiver or a predefined maximum limit has reached. In this way, an μACK sender recovers all transmission errors inside a frame, instead of relying on different retransmission frames.

Figure 1 illustrates a frame transmission in μACK system. In the wide-band feed-forward (FF) channel, the sender sends data symbols to the receiver after a preamble. The receiver, after synchronizing to the preamble, starts immediately a feedback frame to the sender in a narrow-band feedback (FB) channel. Each symbol of the feedback frame acknowledges (or negatively acknowledges) a group of data symbols (GOS) in the feed-forward channel. If the sender gets a NACK, it will mark the corresponding GOS as *lost*. Lost GOSes are re-encoded and appended after the last GOS. As shown in Figure 1, the second GOS contains errors. Then, it is re-encoded after GOS 4, the last group of symbols of the frame. When the entire frame is correctly received, *i.e.*, passed CRC check, the receiver will send back an end-of-stream (EOS) symbol to the sender, which, on receiving EOS, terminates the frame transmission. If the sender does not yet receive an EOS, but also does not have any GOS marked *lost*, it will simply re-encode a GOS from the very beginning (see details in Section3.3).

We now present the μACK feedback design. Based on this design, in Section 3.2 we describe the error and collision detection at the sender. We then describe in details the μACK in-frame error recovery protocols in Section 3.3. We discuss other applications of μACK in Section 3.4. Finally, we finish by discussing several related design issues in Section 3.5.

3.1 μACK feedback

μACK relies on a multi-radio architecture for a receiver to send fine-grained feedback when simultaneously receiving data symbols from the sender. With the increased popularity of wireless communication, multi-radio structure has extensively studied and exploited in previous systems [3]. However, μACK differs from these previous systems in that we integrate both radios into a single board and thus they can be tightly synchronized at micro-second level. Figure 2 shows the system architecture of μACK. Two tightly synchronized radios are deployed at both sender and receiver. The receiver receives data symbols in wide-band FF channel, and based on the decoding results, dynamically modulates and sends feedback symbols in real-time using a narrow-band FB channel. The sender also, in real-time, re-encodes and sends the lost symbols based on received feedbacks.

The key design question here is what shall be the granularity of μACK and how much bandwidth should be allocated to the feedback channel. Ideally, we want to get μACK as fine as possible (*i.e.*, μACK for each received byte) to minimize the overhead of redundant bits. However, too fine granularity μACK will require more bandwidth at the feedback channel to convey this information, which will add overhead to the system. Second, too fine μACK might not be necessary at all since wireless errors are essentially bursty [19]. Bits transmitted within the coherent time usually share the same channel state and are lost in one burst, and therefore may be acknowledged with one μACK. The coherence time depends on the speed of multi-path fading in the wireless channel. In an indoor environment, it varies from around $100\mu s$ (fast fading) to tens of milliseconds (slow fading) [19, 20].

In this paper, we propose to generate one μACK every $20\mu s$, which is much shorter than the coherence time in common indoor wireless channel. This period of time contains five 802.11 OFDM symbols. In theory, the receiver may need to convey only one bit information (*i.e.*, correct or wrong) for every group of symbols it receives. Thus, the feedback channel can be very narrow. For example, the required baud rate the feedback is merely $1/20 = 50KHz$.

Table 2: Symbol encoding in feedback PHY.

Symbol name	Symbol binary ($b_3b_2b_1b_0$)	Chip values
ACK	1100	0111100010
NACK	1001	0011001101
EOS	0110	1100110110

Even with 100% guard-band, it takes around 0.5% overhead of a WiFi $20MHz$ channel. However in practice, we may want more bits in μACK to perform proper coding for reliability. In this paper, we choose the bandwidth of the feedback channel to be $1MHz$. The feedback signal has a width of $500KHz$ and another $500KHz$ spectrum serves as guard-band. We choose $500KHz$ guard-band by referencing to the DECT standard [15], which deploys similar guardband of $576KHz$ with a channel width of $1.728MHz$.

Compared to 802.11, the feedback channel of μACK adds around 5% overhead to the system. But it obviates the need for the original ACK frame and thereby reduces the time-domain overhead. Recall in Section 2, when the frame size is 1500 bytes, the ACK frame normally takes about 8% overhead when the modulation rate is $24Mbps$, which is comparable to our feedback channel overhead. It should also be noted that at higher data rates that involve wider channels (*e.g.*, $40MHz$ channel in 802.11n), the feedback channel overhead decreases reciprocally; while the overhead of ACK frame increases (*e.g.*, 20% with 300Mbps rate) due to shorter transmission time of data symbols [17].

μACK uses very robust code and modulation scheme to ensure high reliability of the feedback. The symbol time of the feedback is equal to the duration of a GOS, which is $20\mu s$. Given 500KHz signal bandwidth, each feedback symbol can be coded into 10 chips. In current design, we map four bits to one feedback symbol and these bits are encoded into 10 chips, each of which is modulated using basic differential binary phase-shift keying (DBPSK). Table 2 summarizes the three feedback symbols currently defined in our system. The remaining symbol values are reserved for future extensions. The entire feedback frame (consisting of smaller ACKs for the data symbols) is preceded with a $20\mu s$ synchronization symbol that allows the sender to detect and synchronize to the feedback frame. The SYNC symbol value is 0x2FF.

3.2 Error detection

μACK requires the receiver to reliably detect erroneous symbol groups to generate real-time fine-grained feedbacks. This partial packet detection problem has been well studied in previous work [8, 12]. In [8], Ganti *et. al.* proposed to split the frame into fragments and insert a CRC check sequence for each fragment. The overhead to transmit these check sequences increase proportionally with the number of fragments in a frame. Alternatively, Jamieson, *et. al.* [12] proposed to use physical layer hints to classify correct or erroneous symbols. These PHY hints usually use the soft-output of the channel decoder. While PHY hints save the overhead of checksums, in our experiments, we find the PHY hint based classifier becomes less reliable when the SNR is just enough to support the modulation rate on a wireless link. Figure 3 shows such an example. The PHY hint in this experiment is the soft-output of Viterbi decoder [11] [1] and the modulation rate is 24Mbps. Figure 3 plots the cumulative fraction of soft-outputs for both correct and erroneous symbols. We see that when the link SNR is 12dB high, using

[1]The output is quantized log-likelihood (LLR) of a correct trellis path in Viterbi decoder.

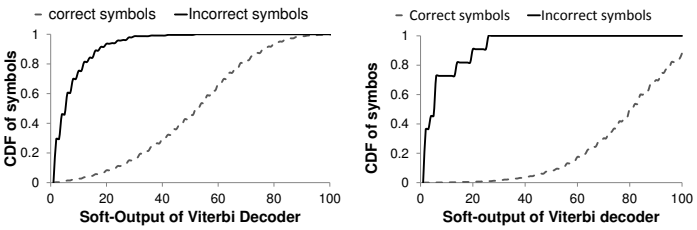


Figure 3: CDF of Viterbi soft-outputs in two SNR settings. The modulation rate is 24Mbps. (a) SNR = 10dB; (b) SNR = 12dB.

PHY hints can detect both correct and erroneous symbols reliably (Figure 3(b)). But when the link SNR reduces slightly to 10dB (just enough to support 24Mbps), the PHY hint cannot easily separate erroneous symbols from the correct ones, and it becomes a difficult tradeoff between higher false negative (choosing a lower threshold) or higher false positive (by choosing a higher threshold). This is expected, since the PHY hints are generally good statistical estimations of error probability for a large amount of symbols [20]. However, they may fall short in reliably predicting correctness of specific set of symbols, especially when the probability of correct and erroneous symbols are comparable at marginal SNR.

Unfortunately, it is not uncommon for a wireless link to work at this marginal SNR regime, as modern rate adaptation mechanisms tend to choose the highest modulation rates for better efficiency. Consequently, in this paper, we allow the sender explicitly embed a CRC checksum along each GOS to facilitate reliable error detection at the receiver.

However, embedding an extra checksum would add extra packet overhead. To avoid this overhead, we exploit a side-channel that has not been used by existing 802.11 OFDM PHY. Current 802.11 PHY deploys four pilot subcarriers in each OFDM symbol, each of which transmits dummy bits. These dummy bits carry no information and are used by the receiver to track channel changes [13]. We argue that such a design, although simple, is not efficient. In this paper, we propose to modulate one bit information on each pilot subcarrier without significantly reducing the channel tracking performance. Specifically, let $P_{i,j}$ denotes the known pilot sequence for symbol i on pilot subcarrier j. Then, instead of inserting $P_{i,j}$, the sender sends $P'_{i,k} = c_k P_{i,j}$, where $c_k = \{1, -1\}$ is a differentially coded binary data. The receiver, however, decodes c_k first, before performing normal pilot tracking. To decode c_k, the receiver uses the pilot value of the previous symbol, $P'_{(i-1),j}$, as a reference, just like a differential demodulation. A group of c_ks can further be protected with error-correction codes, and small number of bit errors may not affect the correctness of c_k. Once c_k is decoded, the receiver can recover $P_{i,j}$ and feed it to normal pilot tracking algorithms. Since in this scheme, we need to detect c_k first, we name our approach as *decision-directed pilot tracking* (DDPT).

We briefly present some reasons on why DDPT does not degrade tracking performance compared to original dummy pilots. We evaluate DDPT in detail in Section 6. First, we note that once c_k is successfully decoded, DDPT is just equivalent to original dummy-bit pilot tracking (DBPT) scheme. This is easy to see, as once c_k is decided, the original pilot value $P_{i,j}$ can be directly recovered. Second, it is reasonable to use a previous received pilot value as a reference to demodulate c_k, since the wireless channel holds stable during the coherence time, which is from $100\mu s$ to several milliseconds for indoor environment like WiFi. The symbol period is about 4 μs which is much shorter than the channel coherence time in our setting. Third, even if there is a sudden interference causing

erroneous detection of c_k, which is BPSK modulated with proper channel coding, we note that in this case the original pilot tracking algorithm may also not perform well, since the interference may already corrupt the tracking results. Therefore, the symbols, in either DDPT or DBPT, suffer a strong interference and fail to decode anyway. Finally, to prevent the decision errors from propagating, we perform DDPT only within a GOS. For the first symbol in a GOS, we always insert normal pilot bits, while modulating information only on the remaining four symbols.

With DDPT, μACK can embed up to 16 bits on the pilot subcarriers of a GOS. In this design, we use a simple Hamming (16,11) code to protect this side-channel. There, we have eleven information bits. Ten bits are used to encode a CRC-10 checksum, which is used in B-ISDN and ATM networks. The other bit is used to indicate whether or not the GOS contains retransmission metadata (detailed in next section). We note that the Hamming code is slightly weaker than the 1/2 code used for 6Mbps rate in 802.11, and may have higher bit error rate (Section 6.1.2). In our future work, we will investigate a better coding scheme for the pilot side channel.

Upon decoding a GOS, the receiver computes a CRC checksum of the decoded bits and compares it to the one embedded in the pilot subcarriers. If they match, the receiver sends out a positive μACK; otherwise it sends a negative μACK.

3.3 In-frame recovery protocol

Based on the error detection and feedback mechanisms discussed earlier, μACK can perform in-frame recovery of erroneous symbols. The in-frame recovery protocol works in the following way.

After detecting a preamble, the receiver starts to transmit the synchronization symbol on the feedback channel. Then, for each GOS it receives, the receiver will send a μACK or μNACK based on the correctness of the GOS. All correctly received data are kept in an assembly buffer. If all received data pass the frame CRC-check, the receiver sends an EOS symbol to notify the sender. If after that, the receiver continues receiving GOS, possibly due to the corruption of the EOS symbols, it simply returns another EOS.

Based on the μACKs received, the sender puts the negatively acknowledged bytes into a retransmission queue. After the last GOS is sent, the sender fetches all data from the retransmission queue and starts re-coding them into *retransmission GOS*(RGOS). RGOSs are sent directly after the last GOS of the frame. Since the data in the retransmission queue are no longer continuous, the first RGOS should contain retransmission metadata to identify the corresponding symbols at the receiver. The RGOS containing the retransmission metadata is marked by the *meta-bit* on its pilot side-channel. The format of the retransmission meta-bit is simple. It contains at least 6 bytes, as shown in Figure 4. The first field, *number of symbols*(10-bit), indicates how many of symbols are re-encoded. The field *size* (6-bit) indicates the number of retransmission block entries in the metadata. Each block entry is 32-bit, including 16-bit start position and 16-bit length. In our current implementation, the size of this metadata header cannot exceed one GOS. For example, if the frame is modulated with 6Mbps, the metadata may contains only 2 entries; while for 54Mbps, the metadata header may hold up to 26 entries. If there are more retransmission blocks that cannot be fitted into one metadata header, multiple metadata headers may need to be added.

All acknowledged data are then removed from the retransmission queue. After one round of RGOS, if there is still any data remaining in the queue, the sender will start a second recovery round. A rare, but possible, scenario is that there is no data remaining, and the sender has not yet received an EOS. For example, there might be a hash collision in CRC-10, such that the receiver mistakes an erroneous GOS as a correct one, and returns a positive μACK. Therefore, to handle this corner case, the sender has to mark all data as lost and retransmit every byte in following symbols.

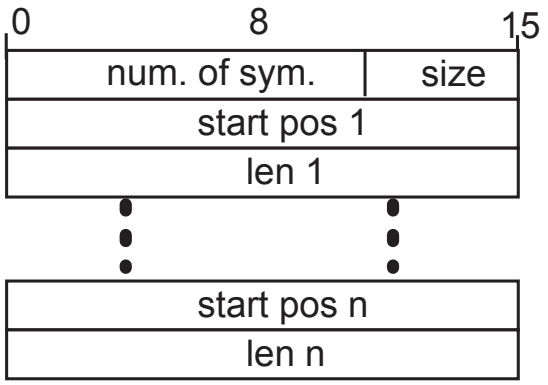


Figure 4: The format of retransmission metadata.

The sender will terminate transmission if it receives an EOS from the feedback channel, or an upper bound of symbols have been transmitted. We set this upper limit to prevent the sender from occupying the channel for an unreasonably long period (Section 3.5). In this paper, we set this upper limit as three times the frame size. The sender will also abort transmission and retry after a backoff time if a significant number of erroneous GOS have been detected ($\geq 50\%$), which usually indicates a persistent collision has occurred.

3.4 Other applications of μACK feedback

Beside the error recovery, the tightly synchronized μACK feedback can also improve the wireless network in following two ways.

Collision detection and early backoff. Using μACK, a sender can detect collisions before the entire transmission is complete. If the sender does not receive the first few μACK symbols, this means the preamble was not likely received by the receiver, which usually implies a collision may occur. Therefore, the sender can immediately stop its transmission, and retry the frame after a backoff time.

Hidden terminal mitigation. μACK can also mitigate the hidden terminal problem without relying on explicit RTS/CTS handshakes, which adds significant overhead. In μACK system, a sender can simply monitor both FF and FB channels. Since the receiver constantly continually sends the μACKs during the data transmission, the feedback can be viewed as a busy tone channel (similar to DBTMA [6]). The contender will defer its transmission once detecting the μACK feedback. Finally, μACK can also help to reduce exposed terminals following the similar heuristic in [6].

3.5 Discussion

We now briefly touch upon some factors in the design of the μACK mechanism.

When to send μACK sync symbol? The receiver should send back synchronization symbol in feedback channel as soon as it detects a frame for it in the data channel. However, in current 802.11 design, the destination address is embedded in the MAC header. That means the receiver can send μACK only after the first GOS has been decoded, adding a large additional delay. Therefore, we propose the extend current PLCP header by an OFDM symbol to store a physical layer address of the receiver. In this way, the receiver can start μACK feedback right after the preamble of the data frame. We note that PHY layer addressing has been previously exploited

in [16,23], and we can use similar approach to dynamically allocate them inside a wireless network.

Range Mismatch: An important requirement is to ensure that the FB channel has a similar range as the FF channel. Otherwise, the hidden/exposed terminal problem will become worse. To ensure the same range, both the channels are in the same band, i.e. either 2.4 GHz or 5 GHz. Furthermore, we leverage prior work on channel widths [4] to adjust the transmit power of the FB channel such that its range matches the FF channel.

Rate Anomaly: Packet fairness of IEEE 802.11 hurts the performance of high data rate nodes in the presence of low data rate nodes [22]. μACKs can make the situation worse since a transmission with large number of bit errors will effectively increase its packet size and occupy the medium for a longer period of time, thereby hurting the performance of transmissions over completely reliable links. To solve this problem, we (i) limit the maximum time a node can occupy the medium including retransmissions, and (ii) reduce the probability of a node that just occupied the medium for a long time from immediately regaining access, similar to the technologies proposed in [22].

Rate Adaptation: Using μACK, a sender can get a good estimate of the BER on the link at any time. Therefore, it is possible to dynamically pick the best rate for each symbol inside a frame. We leave this as our future work.

FB channel allocation: In current two-radio implementation of μACK, we statically assign a narrow FB channel for every FF channel. However, in the future, the frequency of the FB channel in a wireless network may be dynamically selected to avoid potential noisy channels. All FB channel may be allocated to a specific portion of spectrum band. For example, in US, the 11 MHz of FB spectrum (for the 11 channels) can be allocated in the unused channels 12 and 13 of IEEE 802.11b in 2.4GHz band. This spectrum is only available for low-power operation, but given that our ACK is low bandwidth, and hence lower power, we expect to be within the FCC regulations for these channels. Alternatively, we may split a portion of existing 802.11 channel for μACK FB use. For example, we may allocate the upper (or lower) 1MHz of 20MHz WiFi channel to feedback, while the remaining 19MHz spectrum is used for FF data communication. Finally, we note that in the future, with full-duplex technology [5], we expect μACK may have single radio designs and the feedback can be sent using the same frequency as the FF channel.

Frame duration field setting: In 802.11 standard, the MAC header contains a duration field that records the expected frame transmission time plus the ACK. A contender after decoding the field will defer according to the value in this field to avoid possible collision to the ACK frame. For μACK, since the sender may dynamically pad the frame with retransmission symbols, the transmission duration may not be known before hand and cannot be accurately set. However, we note that μACK does not need this duration field for correct protocol behavior. This is because of the following two reasons. First, μACKs are sent in a different feedback channel. Therefore, it is safe for a contending sender to pick up the medium immediately after the FF data channel is sensed idle. Second, as discussed in Section 3.4, μACK can effectively detect collisions and mitigate hidden terminals using the feedback. Therefore, it does not need the traditional RTS/CTS handshake, which further relies on this frame duration field to reserve channel time.

4. ANALYTICAL STUDY

We build on the model presented in Section 2 and analyze the performance of the μACK technique. We also compare its performance to Wi-Fi and the most closely related work, PPR [12].

Model: Suppose the symbol error rate is e_s and a frame contains N_s symbols. Assuming the OFDM symbols are independent, the frame error probability is $e_f = 1-(1-e_s)^{N_s}$. To compare different techniques, we define *retransmission overhead* (RO) as the fraction of additional time to successfully transmit a frame.

$$\begin{aligned} RO &= (T_{ReTx} - T_{data})/T_{data} \\ &= T_{ReTx}/T_{data} - 1, \end{aligned}$$

where T_{ReTx} is the time taken to successfully deliver the packet at the receiver including retransmissions. So, the RO is zero if there is no loss, and greater than zero if there is any retransmission. In our model, we assume the sender will persistently retransmit the lost packet until it is successfully received. We note this is only an approximation to practical systems that may only retry up to a a maximum number (*e.g.*, four times in 802.11). A complete model that considers this maximum retransmission number will be our future work.

Wi-Fi: The expected number of frame retransmissions for Wi-Fi is:

$$\begin{aligned} K_{wifi} &= 1*(1-e_f) + 2*e_f(1-e_f) + ... \\ &\quad +k*e_f^{(k-1)}(1-e_f) + ... \\ &= 1/(1-e_f) \end{aligned}$$

Therefore, extending the formulation of Section 2:

$$T_{ReTx} = T_{data}(R) + \sum_{i=1}^{K_{wifi}} T_{data}(R_i).$$

where R_i is the data rate used for the i^{th} retransmission packet.

PPR: We assume that there is no retansmission aggregation in PPR for latency reduction. Also, we do not model the dynamic programming algorithm that PPR uses to compute the optimal trunks to retransmit. Modeling such a dynamic algorithm is non-trivial. So we leave it as our future work. Here, we just try to build an approximate model: We assume PPR retransmits only erroneous bits and we don't consider the overhead of CRC-checksum for the runs of good bits. Clearly, our approximate model captures an performance upper bound of PPR.

Based on this simplification, we can compute the size of the k^{th} retransmission frame, s_k, in PPR as

$$s_k = e_s s_{(k-1)}.$$

The frame error rate of the k^{th} retransmission is:

$$e_f(k) = 1 - (1-e_s)^{(s_k)}.$$

Therefore,

$$T_{ReTx} = T_{data}(R) + \sum_{k=1}^{\infty} \left(\prod_{i=0}^{k-1} e_f(i) \right) T_{data}^k(R_k),$$

where $T_{data}^k(R_k)$ is the air-time to send the k^{th} retransmission (s_k symbols) with rate R_k.

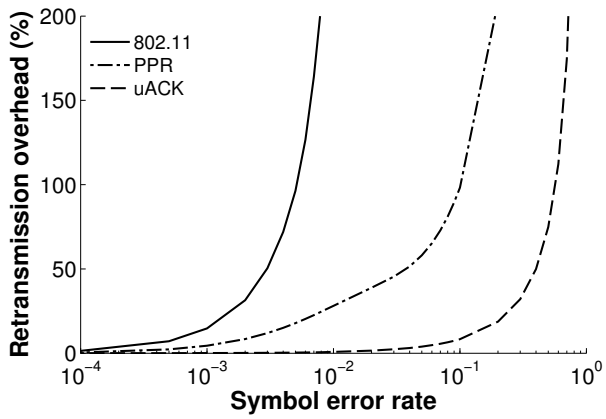


Figure 5: Retransmission overhead versus the symbol loss rate on the wireless link. The packet size is 1500B and the data rate is 54Mbps.

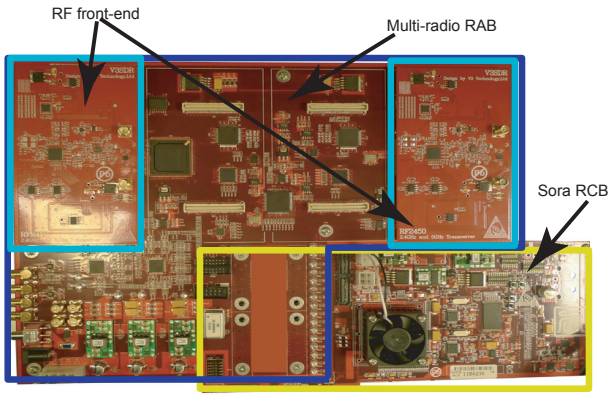


Figure 6: Multiradio RAB for Sora. It can connect to up to four RF front-ends. Two RF boards have been plugged on the board, and there are two slots empty.

μACK: In our system, the erroneous symbols are directly appended at the end of the frame. The number of retransmitted symbols is

$$N_{rx} = N_s \cdot \sum_{i=1}^{\infty}(e_s^i) = N_s e_s/(1-e_s)$$

Let l is the average burst length of erros. Then, the expected frame size in μACK is

$$N_{data} = N_s + N_{rx} + \lambda N_{rx}/l,$$

where λ is the metadata overhead to describe a burst loss in μACK (Section 3.3). Finally, we can derive the expected air-time to send a frame using μACK,

$$\begin{aligned} T_{ReTx}(R) &= t_{CW} + t_{DIFS} + t_{DATA} + t_{\mu ACK\_delay} \\ &= 72\mu s + 28\mu s \\ &\quad +(20\mu s + t_{symb}\lceil N_{data}/(4R)\rceil + 6\mu s) \\ &= T_{data}(R) + t_{symb}(\lceil N_{rx}(1+\lambda/l)/(4R)\rceil) \\ &\quad -T_{ack} + t_{\mu ACK\_delay}, \end{aligned}$$

where $T_{ack} = t_{SIFS} + t_{ACK}$ is the ACK overhead and $t_{\mu ACK\_delay}$ is the latency of μACK feedback.

Figure 5 presents the quantitative results of 802.11, PPR, and μACK under different symbol error rates. When the symbol error rates are low, the link is very reliable and the retransmission overhead is also low. As the link becomes lossy, the retransmission overhead of 802.11 increases exponentially and soon hits a wall at symbol error rate of 10^{-2}, meaning that no matter how many times it retransmits, the packet cannot be delivered. PPR reduces the retransmission overhead by resending only the lost bits. Therefore, the retransmission overhead increases slower than 802.11, but the header and contention overhead is still high. μACK, however, maintains a very low retransmission overhead even in very lossy environments, *i.e.*, the symbol error rate is as high as $1/2$.

5. IMPLEMENTATION

5.1 Platform

We have implemented μACK based on the Sora software radio platform [18]. To fully support μACK, we extended Sora in following two ways:

Multi-radio RAB. We built a new Sora RAB[2] that is able to connect to four RF front-ends as shown in Figure 6. These four RF front-ends are synchronized using the same oscillator, but can be independently programmed. For example, they can be configured in different spectrum bands with different channel widths. Or they can be in different transmitting or receiving modes. Each RF front-end exposes an abstract radio object in Sora system and the software radio applications can separately control each of them.

Streaming Tx interface. Current Sora SDK supports only frame-based transmission control interface. A frame of modulated signal should be entirely transferred into the RCB memory first, before it can be transmitted over the RF front-end [18]. This two-phase Tx method has a latency that is unacceptable for μACK. In this work, we extend the Sora system to support a new streaming Tx interface. Instead of downloading the entire modulated signal, the system can start transmitting a block of samples, while at same time fetching the next sample block from PC memory. Therefore, the modulation process and the signal transmission can be pipelined on Sora. In current implementation, the sample block contains 1.4 μs samples. This way, we can achieve dynamic modulation in real-time at the granularity of OFDM symbols.

5.2 μACK implementation

μACK is implemented as a user-mode SDR application based on Sora UMX API [1]. The program contains two *exclusive threads (ethread)*. One *ethread* runs the standard 802.11 OFDM PHY (SoftWiFi); the other one runs the feedback PHY as described in Section 3.1. We modified about 250 lines of code to SoftWiFi to support the streaming Tx interface and the pilot side-channel. The feedback channel occupies 1MHz frequency, within which 500KHz is used for communication and the rest serves as guard-band. Since the feedback PHY is very simple and contains limited number of symbols (three valid symbols plus the sync symbol), we build a lookup table to modulate μACK feedback and we apply the maximum-likelihood decoding to a received feedback symbol. In other words, the feedback receiver simply compares the demodulated symbol to each valid symbols and pick the one with the minimal Hamming distance.

[2]Radio adaptive board, which is used to connect a 3rd party RF board to Sora RCB.

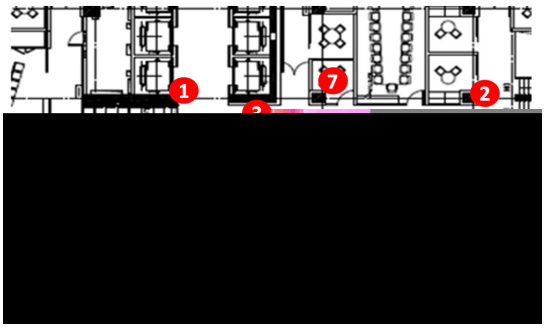


Figure 7: The software radio testbed layout: there are total 9 Sora nodes located in offices and lobby in an indoor environment.

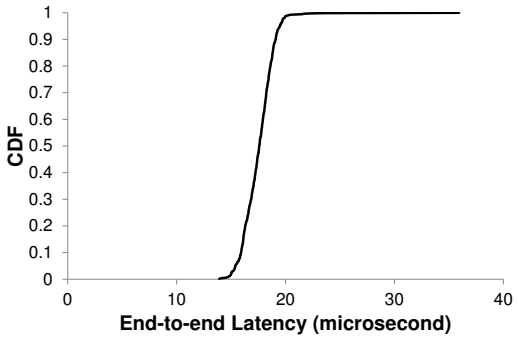


Figure 8: End-to-end delay of μACK feedback.

6. PERFORMANCE EVALUATION

We evaluate μACK in this section. We begin in Section 6.1 the micro-benchmark of our system implementation on the software radio platform. Then, we perform trace-driven simulation to evaluate μACK in a relatively large networking setting and compare μACK to original 802.11 as well as PPR [12].

Method. The micro-benchmarks are mainly conducted in a controlled environment. We connect two nodes' RF front-end using cables and manually add attenuation to create different SNR on the link.

We have built a test-bed contains 9 Sora nodes. Figure 7 shows the layout of our testbed. The Sora nodes are placed in offices and lobby in an indoor environment. The channel on which we collect data is centered at 2422MHz. This is the least busy channel in our office building, but still has uncontrollable interference from neighboring APs. Since we use software radio, we can log the decoding status of each symbol as well as the exact time a frame is received or transmitted. The large PC memory can easily hold a few minutes of frame logs without loss before writing to hard disk. We verify the links between nodes by selecting one node in turn as the sender to broadcast to all other nodes. We find the most links have a packet loss rate between $10 \sim 40\%$. Few links have very good quality to support 48Mbps data rate with few losses. There are also hidden terminals in our testing network. In average, each node may find one hidden node in the network.

We try all data rates on each link and identify the best modulation rate that has the best throughput on the link. We use this best modulation rate for both original frames and retransmissions on the link. Then, we pick up one node as a receiver and all other nodes send a saturate unicast traffic to it. All senders use the best modulation rate measured before. All sending and receiving frames are logged on senders and the receiver. We measure 10 runs for each receiver; each run last 30 seconds. Then, we change the receiver and repeat the measure until all nodes in the network have been covered.

6.1 Micro-benchmark

We first evaluate the end-to-end latency of μACK. This latency defined here is the period between the time when the last sample of a GOS is transmitted and first sample of a corresponding μACK that is received by the sender. This latency includes the demodulation/decoding of the GOS, modulation the acknowledgment, as well as the delay incurred by the hardware platform. Figure 8 shows the distribution of the end-to-end latency measured at the sender. We can see that our system can quickly generate and return μACK in a short delay. The mean value is only about 17.5 μs.

Table 3: Breakdown latency of μACK feedback (μs).

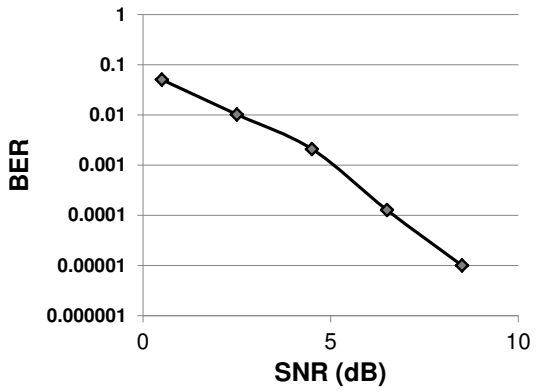
Viterbi decoding	μACK modulation	Hardware
7.5 (5.82, 8.06)	1.96 (1.4, 2.23)	9.103 (8.22, 10.5)

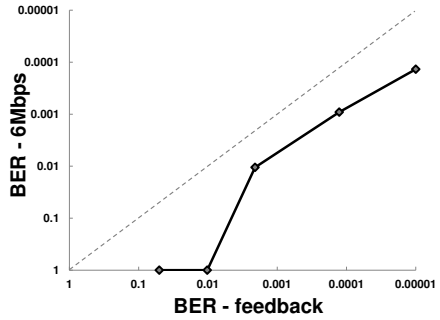
To better understand the latency, we break down the latency of each component in Table 3. The Viterbi decoder introduces about $8\mu s$ delay. This is reasonable. Since the Viterbi algorithm requires certain trellis depth before it can trace-back the results [3], the decoder needs to finish processing symbol $(i+1)$ to output symbol i. Consequently, it adds the processing delay of 2 symbol time. The modulation of μACK is trivial, less than 2 μs. This is because we have optimized the modulation process using lookup tables. Thus, this latency is only the time needed to copy samples of one feedback symbol. The hardware platform contributes the most latency in our system. Since we use a software radio platform, both sending and receiving a symbol will introduce a delay about $1.4\mu s$ to pass the PCIe bus. Therefore, there will add up to 5.4 μs in the end-to-end latency as the samples need to pass the PCIe bus for four times - sending and receiving data/μACK symbols at both sender and receiver. The Sora Fast Radio Link (FRL) used to connect the RCB and the RF front-end contributes the rest latency. Together, the total hardware latency is about 9 μs. We note this hardware latency can be significantly reduced if we implement μACK in silicon.

6.1.1 μACK feedback

Second, we evaluate the reliability of our feedback channel. We perform this measurement in a controlled environment. We connect the two nodes with wires and manually adds attenuation to control the SNR value. Figure 9(a) shows the μACK feedback error rate versus SNR curve of our feedback PHY as discussed in Section 3.1. When SNR is as low as 2.5dB, our μACK feedback has a low error rate of 10^{-2}. The error rate quickly drops to 10^{-4} when SNR is about 6.5dB. Figure 9(b) compares the reliability of μACK feedback channel with the lowest data rate (6Mbps) of 802.11. Each data point presents the BER of μACK feedback and 802.11 6Mbps rate in the same wireless link condition. From the figure, we can conclude that the feedback PHY is much more reliable than the lowest data rate used in 802.11, and adding such a fine-grained feedback will not significantly increase the end-to-end symbol error ratio.

[3](The depth is usually no less than $5K$, K is the constraint size of the convolutional encoder.





(a) (b)

Figure 9: The reliability of μACK feedback channel. (a) BER of feedback PHY versus SNR. (b) Comparision of μACK feedback with 802.11 OFDM 6Mbps rate.

6.1.2 DDPT performance

In this experiment, we evaluate the performance of DDPT (Section 3.2). Again, we perform the experiment in the controlled environment where the channel coherent time is relatively large. The sender broadcasts frames to the receiver over the controlled link. The broadcast traffic is 2Mbps with a frame size of 1000 byte. We configure various SNR values on the link. For each data rate, we first send frames without DDPT (normal 802.11 rates), and then we turn on DDPT. We record the frame loss rate at the receiver. Each measure takes 20 second. Figure 10 plots the results. Each data point in the figure represents the frame loss rate we get for a data rate with and without DDPT. In general, we can see the difference is quite small, especially when the data rates are high. With low rates (*i.e.*, 6Mbps and 9Mbps), using DDPT may slightly increase the frame error rate. This is reasonable. Since 6 and 9Mbps rates are already using BPSK modulation, their bit error rate is comparable to DDPT side channel. Therefore, the error rate added by DDPT modulation may be observable. But with high data rates, the data subcarriers are using high modulation/coding mode, and may already become very erroneous while DDPT is still able to track the channel reliably.

Figure 11 quantifies the BER of the DDPT side channel. The performance is comparable to that of 6Mbps modulation rate of 802.11. For example, to achieve BER of 10^{-3}, DDPT requires 8dB SNR, which is slightly higher (around 1dB) than 6Mbps rate of 802.11.

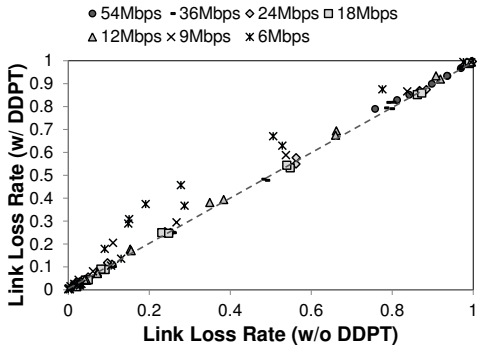


Figure 10: Comparision between DDPT and DBPT.

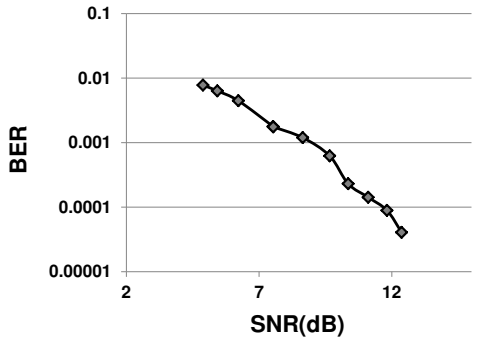


Figure 11: Performance of DDPT side channel.

6.2 Single link performance

We compare the wireless link throughput of three schemes:

1. **802.11g.** This is normal 802.11g that always retransmits the entire frame.

2. **PPR.** We implement PPR [12]. As mentioned earlier in Section 3.2, the PHY hits may work unreliably when the link SNR marginally supports the chosen data rate. For a fair comparison, we also use the pilot side-channel to transmit CRC checksum for PPR as well. After a partial frame is received, the receiver will compute a feedback in an ACK frame and the sender, upon receiving the ACK, send the retransmission in a new frame.

3. **μACK.** μACK exploits the fine-grained feedback from the receiver in a separate channel to re-encode the lost symbols and append them right after the frame. In current implementation, μACK uses more bandwidth – we use total 21MHz spectrum, *i.e.*, 20MHz for forward data channel and 1MHz for μACK feedback. For a fair comparison, we reduce the measured throughput by 5% before we plot it in the figure.

For every SNR setting, we try all data rates for each scheme and we present the result of each scheme with the best modulation rate that yields the highest throughput on the link. For all schemes, we use this best modulation rate for both original frames and retransmissions.

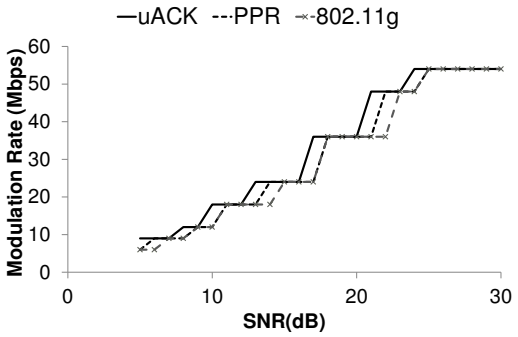


Figure 13: Best rate selected of different schemes at different SNR value.

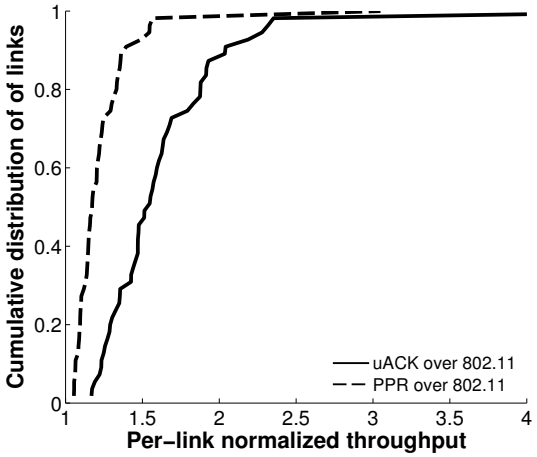


Figure 14: Per-link throughput distribution in the testbed.

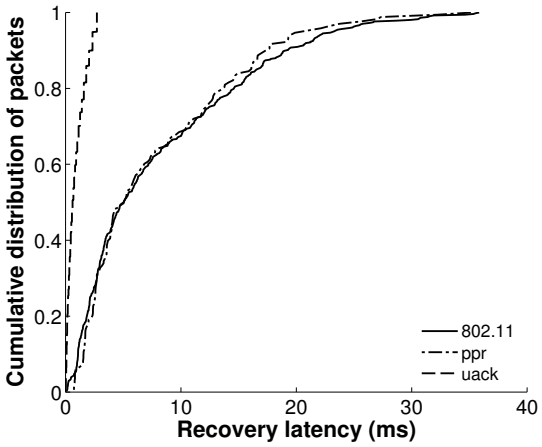


Figure 15: Error recovery latency.

Results. Figure 12 shows the results. When SNR is low, the link is very lossy even though the most reliable rates are applied. Therefore, the retransmission overhead is high for 802.11g. By avoiding retransmission of the redundant data, PPR and μACK can significantly improve the link throughput. With the increase of SNR, the link becomes less lossy, and higher modulation rates can be applied. Therefore, the benefit of PPR also becomes less, since in this case the backoff, headers and ACK dominates the retransmission overhead. They become more severe as the frame size decreases (The bottom row of Figure 12). μACK, however, removes such overheads using a narrow-band feedback channel and thus always gains over other two schemes. Further, as μACK decreases the retransmission overhead, it has better chance to use a higher modulation rate, which gives more throughput gain. Figure 13 shows this effect. We can see that μACK can utilize a higher rate one or two dB earlier than other two schemes. In summary, μACK improves the link throughput by $8 \sim 220\%$ over 802.11g, and $5 \sim 30\%$ over PPR.

During our experiments, we also find PPR sometimes may deliver less throughput than original 802.11g (as shown in Figure 12. The reason is due to PPR's postample. In this situation, the postample adds a fairly amount of overhead, but gains nothing – there is no hidden terminal here. So, we remove the postample and rerun the experiments, listed as *PPR-no post* in Figure 12. PPR-no post always performs better than 802.11g, but still below μACK.

6.3 Testbed experiments

In this section, we present the trace-driven emulation to evaluate μACK performance in a busy network. The trace is collection from a 9-node software radio network as described at the beginning of Section 6. The question we are answering here is what performance gain we can get if we apply μACK in such a network comparing to other schemes, *i.e.*, 802.11g and PPR. The trace-driven emulation is performed as follows: We examine each receiver trace one-by-one. We parse the decoding results of each symbol of each frame. For PPR, once a partial frame is detected, we compute the feedback message that should be embedded inside an ACK. Then, we adjust the ACK size accordingly, and treat the next frame as the retransmission. We further adjust the size of the frame according to the PPR protocol and determine the delivery status based on this new frame size. For μACK, we re-encode lost symbols after the frame. We also refer to the recorded results of the next frame to decide the decoding status of these retransmitted symbols. The backoff window are carefully adjusted according to the decoding results of the previous packet.

Figure 14 shows the per-link throughput distribution of our testbed. The throughput is normalized to 802.11g. We see that PPR has up to 60% (median 20%) throughput gain compared to 802.11g. This gain is much larger compared to a controlled environment in Figure 12. This is because in our test-bed network, the loss is much more prevalent due to the interference from other transmitters. μACK can achieve up to 140% (median 60%) throughput gain over 802.11g and up to 60% (median 30%) gain compared to PPR.

Figure 15 illustrates the delay distribution to recover a lost packet. The recovery delay is defined as the time between the first transmission and the finally delivery of packet. We assume the sender uses up to 4 times retransmission for both PPR and 802.11. We can see that both PPR and 802.11 has large recovery delay (up to 40ms!). This is because each retransmission packet should go entirely contention process to gain the access to the channel. In a busy network, this contending period can be very long. μACK, however, does not give up the channel and all errors are recovered inside the frame. So the recovery delay is very low (up to 4ms). This property is especially beneficial to real-time applications, like real-time conferencing or gaming, that requires very short delay in packet delivery.

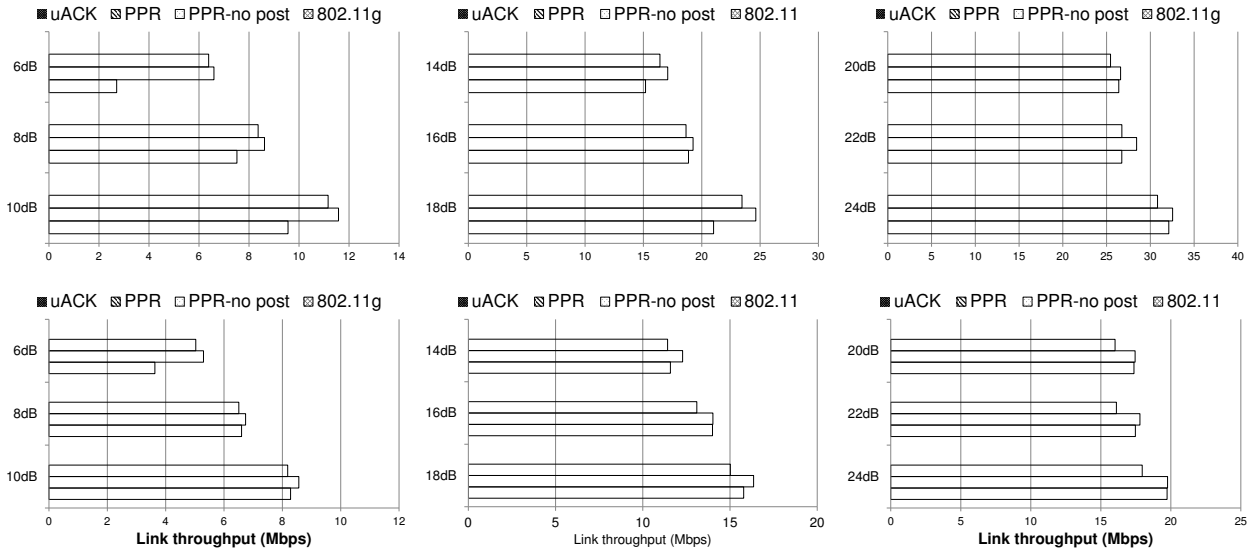


Figure 12: Link throughput of three schemes under different SNR. Top row: packet size is 1500B. Bottom row: packet size is 500B. Left: Low SNR regime. Middle: Middle SNR regime. Right: High SNR regime. X-axisY-axis shows the SNR (dB).

7. RELATED WORK

There is a large body of work on improving the reliability of wireless communication. Due to lack of space we only touch upon the most closely related work in this section, and refer the reader to [19] for more background.

ARQ is a well-known technique to recover from losses, where packets without ACKs are retransmitted. Hybrid ARQ [14] reduces the retransmission overhead using FEC, in addition to ARQ. Type I Hybrid ARQ resends the entire packet (along with FEC and error detection (ED)) when the packet is lost. Type II Hybrid ARQ reduces the ARQ overhead when the medium is mostly loss-free. Packets include ED but no FEC the first time they are sent. Retransmissions include both FEC and ED. Our μACK mechanism is complementary to the 802.11 Hybrid ARQ. Once the mapping from the data including FEC and ED is done to symbols, we only modify the ACK mechanism to be on a finer granularity, i.e. per symbol, instead of per-packet.

The most closely related work to μACKs is PPR [12], which we have referred and compared to in various sections in this paper. PPR reduces retransmission overhead by eliminating redundant bits. We reduce the overhead further, by not requiring separate packet transmissions (and the consequent overhead) for retransmissions.

CSMA/CN [16] utilizes the feedback to detect collision. It uses soft-phy hint to detect collisions. Once a collision is identified, a notification is sent back in the same frequency band, assuming the sender has full-duplex radio. Then, the sender may abort the transmission earlier. μACK has significant different design compared to CSMA/CN. First, μACK utilizes a narrow-band, dedicated feedback channel with little overhead; while the notification in CSMA/CN is a pseudo-random wide-band signal. Second, CSMA/CN notification conveys only one bit information (collision or not); while μACK feedback can contain multiple bits information. As a consequence, μACK design is much more powerful. It can not only detect collisions (similar to CSMA/CN), but also be able to perform in-frame retransmission to improve the error recovery efficiency. Further, the μACK feedback is continuous during the transmission of the data frame, and therefore can serve as an extended busy-tone to mitigate hidden and expose terminal problems.

Using separate channel to send acknowledgments is common in FDD wireless systems, *e.g.*, FDD LTE [7]. However, in such systems, the ACK is still at the frame level. For example, LTE NodeB will send an ACK/NACK subframe in the downlink channel after it receives a sub-frame in the uplink channel from a mobile node. In contrast, μACK sends back feedback symbols during the transmission of the data frame that allows in frame adaption (*e.g.*, symbol retransmission). Further, LTE is a tightly synchronized wireless system and only applied in licensed spectrum bands. However, μACK is designed for a random access wireless network, like WiFi, and is suitable for unlicensed spectrum bands.

Rate control algorithms try to make transmissions more reliable without losing capacity by appropriately adjusting the rate. For example, SoftRate [20] uses the SoftPHY hints to determine the best rate to send the packet. Using μACKs, a sender can get more immediate feedback about symbol error rate, and can in fact adapt the rate in between the transmission, i.e. as and when it receives the μACK for the previous GOS. Another recent rate adaptation algorithm is Strider [10], which proposes using rateless codes, which can significantly improve the reliability of transmissions. Similar to SoftRate, μACKs can improve the performance of Strider, as it can adjust the codes on a GOS time scale instead of the entire packet.

Another body of related work uses control channels. For example, RI-BTMA [21] and DBTMA [6] use busy tones on the control channels to resolve contention and hidden terminals, respectively. Although other work has also looked at offloading control traffic to the control channel, we believe μACK is the first system that integrates the control channel for in-line acknowledgment of symbols as the packet is being transmitted.

8. CONCLUSION

In this paper, we quantify the overhead of frame retransmissions, and show that the overhead is significant in a lossy wireless network. We breakdown the overhead from duplicate headers, con-

tention as well as redundant bits, and propose a new in-frame retransmission scheme using μACKs. Instead of waiting for the entire transmission to end before sending the ACK, the receiver sends smaller μACKs on every few symbols, on a separate narrow feedback channel. Based on these μACKs, the sender only retransmits the lost symbols after the last data symbol in the frame, thereby adaptively changing the frame size to ensure it is successfully delivered.

We have implemented μACK on the Sora platform. Experiments with our prototype demonstrate that μACK can significantly reduce the retransmission overhead. Therefore, the sender can aggressively use higher data rate on a lossy link, which further improves the overall network efficiency. We believe that μACK is a powerful idea and has other applications beyond error-recovery. The feedback mechanism developed in the context of μACK can be used by a variety of future wireless protocols.

9. ACKNOWLEDGEMENTS

The authors would like to thank Dongliang He for his initial work on the frame delivery prediction based on soft-viterbi decoder in MSRA. His work has inspired the error detection scheme in μACK. The authors also thank Prof. Dirk Grunwald and anonymous reviewers for their valuable comments and suggestions to improve the paper.

10. REFERENCES

[1] The sora manual version 1.5. *http://research.microsoft.com/apps/pubs/?id=144847*.

[2] *ANSI/IEEE Std 802.11, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*. IEEE Press, 1999.

[3] P. Bahl, A. Adya, J. Padhye, and A. Walman. Reconsidering wireless systems with multiple radios. *SIGCOMM Comput. Commun. Rev.*, 34:39–46, October 2004.

[4] R. Chandra, R. Mahajan, T. Moscibroda, R. Raghavendra, and P. Bahl. A case for adapting channel width in wireless networks. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 135–146, New York, NY, USA, 2008. ACM.

[5] J. I. Choi, M. Jain, K. Srinivasan, P. Levis, and S. Katti. Achieving single channel, full duplex wireless communication. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, MobiCom '10, pages 1–12, New York, NY, USA, 2010. ACM.

[6] J. Deng and Z. Haas. Dual busy tone multiple access (dbtma): a new medium access control for packet radio networks. In *Universal Personal Communications, 1998. ICUPC '98. IEEE 1998 International Conference on*, volume 2, pages 973 –977 vol.2, oct 1998.

[7] J. S. Erik Dahlman, Stefan Parkvall. *4G: LTE/LTE-Advanced for Mobile Broadband*. Academic Press; 1 edition, 2011.

[8] R. K. Ganti, P. Jayachandran, H. Luo, and T. F. Abdelzaher. Datalink streaming in wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, pages 209–222, New York, NY, USA, 2006. ACM.

[9] M. Gast. 802.11 Wireless Networks: The Definitive Guide. In *O'Reilly*, 2005.

[10] A. Gudipati and S. Katti. Strider: automatic rate adaptation and collision handling. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 158–169, New York, NY, USA, 2011. ACM.

[11] J. Hagenauer and P. Hoeher. A viterbi algorithm with soft-decision outputs and its applications. In *Global Telecommunications Conference, 1989, and Exhibition. Communications Technology for the 1990s and Beyond. GLOBECOM '89., IEEE*, pages 1680 –1686 vol.3, nov 1989.

[12] K. Jamieson and H. Balakrishnan. Ppr: partial packet recovery for wireless networks. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '07, pages 409–420, New York, NY, USA, 2007. ACM.

[13] H. Meyr, M. Moeneclaey, and S. A. Fechtel. *Digital Communication Receivers: Synchronization, Channel Estimation and Signal Processing*. Cambridge University Press, 1997.

[14] T. K. Moon. *Error Correction Coding*. New Jersey: John Wiley & Sons, 2005.

[15] K. J. Saldanha. *Performance Evaluation of DECT in Different Radio Environments*. Ph.D Thesis, Virginia Polytechnic Institute and State University.

[16] S. Sen, R. Roy Choudhury, and S. Nelakuditi. Csma/cn: carrier sense multiple access with collision notification. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, MobiCom '10, pages 25–36, New York, NY, USA, 2010. ACM.

[17] K. Tan, J. Fang, Y. Zhang, S. Chen, L. Shi, J. Zhang, and Y. Zhang. Fine-grained channel access in wireless lan. In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 147–158, New York, NY, USA, 2010. ACM.

[18] K. Tan, J. Zhang, J. Fang, H. Liu, Y. Ye, S. Wang, Y. Zhang, H. Wu, W. Wang, and G. M. Voelker. Sora: High performance software radio using general purpose multi-core processors. In *6th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2009)*.

[19] D. Tse and P. Vishwanath. *Fundamentals of Wireless Communications*. Plenum Press New York and London, 2005.

[20] M. Vutukuru, H. Balakrishnan, and K. Jamieson. Cross-layer wireless bit rate adaptation. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 3–14, New York, NY, USA, 2009. ACM.

[21] C. Wu and V. Li. Receiver-initiated busy tone multiple access in packet radio networks. In *Proceedings of the ACM SIGCOMM 1987 conference*, SIGCOMM '87, pages 336–342. ACM, 1987.

[22] D.-Y. Yang, T.-J. Lee, K. Jang, J.-B. Chang, and S. Choi. Performance Enhancement of Multi-Rate IEEE 802.11 WLANs with Geographically-Scattered Stations. *IEEE Transactions on Mobile Computing*, pages 906–919.

[23] X. Zhang and K. G. Shin. E-mili: energy-minimizing idle listening in wireless networks. In *Proceedings of the 17th annual international conference on Mobile computing and networking*, MobiCom '11, pages 205–216, New York, NY, USA, 2011. ACM.