The Hare and the Tortoise: Taming Wireless Losses by Exploiting Wired Reliability

Anirudh Badam
Princeton University
abadam@cs.princeton.edu

Dongsu Han
Carnegie Mellon University
dongsuh@cs.cmu.edu

David G. Andersen
Carnegie Mellon University
dga@cs.cmu.edu

Michael Kaminsky
Intel Labs
michael.e.kaminsky@intel.com

Konstantina Papagiannaki
Intel Labs
dina.papagiannaki@intel.com

Srinivasan Seshan
Carnegie Mellon University
srini@cs.cmu.edu

Abstract

Multiple communication channels are common in today's consumer and enterprise networks. For example, a high bandwidth but unreliable wireless network might co-exist with a reliable wired link (EWLANs and neighborhood networks). In this paper, we present a system that uses this reliable wired communication channel to boost the bandwidth of the lossy wireless link. Specifically, we propose a new, efficient partial packet recovery (PPR) technique and adaptive feedback mechanism specially designed to correct partial packets on an 802.11 wireless network using a wired backhaul. Our initial experiments demonstrate up to a 3x improvement over standalone 802.11 and upto a 30% improvement over existing PPR techniques.

1 Introduction

At many points during the evolution of communications technology, designers have faced the challenge of combining multiple links, with different properties, to achieve high throughput and low cost. Examples range from the hybrid satellite systems of the early 90s (a high-bandwidth satellite downlink augmented with a slow dialup modem uplink), to recent proposals that combine a cheap, fast, and unreliable wireless link with a reliable satellite or dialup link for developing regions [2], and even to the use of wireless in the predominantly ethernet based datacenter [15]. In this paper, we tackle this problem for a "hare and tortoise" link combination: Using a reliable wired link (the tortoise) in an efficient manner to improve throughput on a high-bandwidth but lossy wireless link (the hare). We term such combinations "HTL" (hare-tortoise link) networks.

Prior solutions to this problem mostly fall into two groups: link-selection mechanisms and striping mechanisms. Migration or vertical handoff [22] allows users to use the best single link available to them at a given time. Other link selection mechanisms try to best match a flow and a link based upon the application requirements or characteristics [26]. Striping mechanisms load-balance across the links on a per-flow basis to maximize the bandwidth achieved from the combined links (e.g., FatVap [14]). These systems provide at most the sum of the individual isolated capacities. In contrast, in the HTL scenario, the combined link bandwidth can be increased beyond the sum of the individual isolated capacities—bandwidth of the lossy hare link can be boosted by exploiting the reliability of the tortoise link.

In this paper, we present a new technique, HTPPR, that leverages the reliable tortoise link to increase the effective bandwidth of the lossy hare link. It does so by using the tortoise link to circumvent some of the biggest challenges of using a wireless link: perfectly estimating channel conditions to determine the amount of FEC to apply, providing fine-grained acknowledgments to support ARQ, and ensuring that vital control information is received even though the channel is lossy. HTPPR not only increases the effective bandwidth of the hare link but also minimizes the usage of the tortoise link for sending the vital channel information such that the combined bandwidth increases.

As we describe further in Section 3, HTPPR incorporates prior work on partial packet recovery [24] to effect these gains as a proof of concept. A general solution to this problem must achieve higher combined bandwidth without imposing high latency.

The primary contribution of HTPPR is the design and implementation of an HTL-targeted link bonding mechanism that increases the hare link's bandwidth by efficiently sending wireless channel information over the tortoise link such that the combined capacity increases. As part of achieving this goal, we present the design of a bandwidth-efficient feedback mechanism for the hare link, and a low overhead accurate channel error estimation method and an accompanying multi resolution partial packet recovery scheme. To show that HTPPR can be used practically, Section 4 presents a data transfer application that uses HTPPR to provide high-bandwidth inter-home communication within a neighborhood by combining wireless links between the residences with the homes' reliable wired Internet access links. Experiments with our prototypes show that HTPPR can improve throughput of the hare link by 3x over standalone 802.11 networks (Section 5), and by up to 30% over existing wireless-only PPR mechanisms while increasing the overhead on the tortoise link only by a negligible amount.

2 Making the Most of the Hare

Making optimal use of a wireless link requires two mechanisms: First, sending using an appropriate forward error correction (FEC); and second, efficiently retransmitting data that was corrupted beyond the ability of the FEC. Both techniques need a reliably and timely feedback from the receiver. Hence, increasing the bandwidth of the lossy hare link by sending feedback reliably on the tortoise link is one way of increasing the combined bandwidth of an HTL-link. In the rest of the paper we focus on increasing the hare link bandwidth while minimizing the overhead on the tortoise link such that the combined bandwidth of the links actually increases.

To demonstrate how a small amount of capacity on a highly reliable link could lead to more-than-par gains in the bandwidth of a less reliable wireless link, we utilize many recent key observations for improving wireless throughput – partial packet recovery [13, 17, 24], estimated bit-error rate based wireless bit rate selection mechanism [7, 23] and data-oriented transfer for wireless mesh networks [8]. However, our techniques to obtain benefits on the wireless link while at only a small overhead on the reliable link are unique. We also present new techniques to reduce wireless channel contention by minimizing the number of small packets and by reducing the amount of metadata sent on the wireless channel. Also, we currently focus mainly on the transmission of static content that can tolerate jitter and high latency. In the future, we wish to explore how interactive data (live streaming of video and audio) can also be improved even when the latencies of the hare and the tortoise links are quite different.

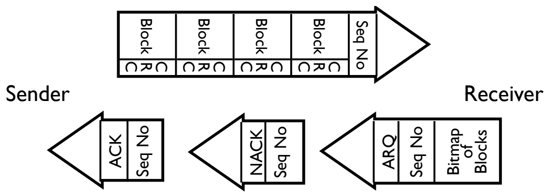


Figure 1: Partial Packet Recovery

Current wireless systems recover from both completely and partially lost packets by retransmitting the entire packet, this strategy is not optimal. Research systems such as ZipTx [17] describe mechanisms to harness partial packets. Our technique, HTPPR, includes a partial packet recovery mechanism specifically designed for HTL links.

Figure 1 depicts an example of a typical partial packet recovery system. Data packets consist of multiple *blocks*. A packet is considered corrupt when any of its blocks fails the block-level CRC check. A packet is considered lost when a new packet arrives with a higher than expected sequence number. For each correctly delivered packet, the receiver transmits an acknowledgment (*ACK*) back to the sender. For each corrupt packet, the receiver sends a retransmission request packet (*ARQ*) that includes a bitmap of erroneous blocks in the packet. Each lost packet generates a negative acknowledgment (*NACK*). The sender processes these ACKs, ARQs and NACKs and retransmits the missing blocks or new blocks as needed. Similar schemes exist for FEC-based PPR approaches.

Wireless-only, multi-round PPR techniques face many challenges. Some spend substantial wireless capacity on feedback and metadata—for example, in ZipTx, ACKs, NACKs and ARQs are exchanged for almost every wireless packet sent. Other systems rely on finer grained information available only to software radios [13, 24]. Feedback and metadata information is even more important than normal data—if the feedback or header data is lost, the whole packet is lost. This metadata must therefore be protected even more strongly by FEC, or transmitted in different packets, consuming additional wireless capacity.

The wireless feedback must help answer three questions: 1) What part(s) of the data has been received; 2) What additional information does the receiver need in order to correct corrupted data; and 3) What bit rate should the sender select to obtain the highest goodput from the channel. Our goal is to answer these questions jointly, providing an efficient feedback mechanism to address the first two, and leveraging the detailed feedback information to perform bit rate adaptation. In this paper, we address these problems by leveraging the reliable tortoise link for sending feedback and metadata. This creates its own challenge in balancing between the two links: The design must operate such that the overhead on the tortoise link is minimal so that the combined bandwidth actually increases.

3 HTPPR Design

The first question for HTPPR is how to use the links in the system. Our design arises from two properties of an HTL link and of the PPR techniques we use:

- Overhead on "tortoise" link must be minimized;
- Metadata sent via the "hare" link must be protected against loss, increasing its overhead;

We first start by describing the PPR mechanism that harnesses partial packets on the hare link by sending vital metadata and feedback reliably on the tortoise link. Using a reliable link for the metadata and feedback means that HTPPR does not need to apply extra error protection to it—in short, it is a more valuable use of the reliable link's capacity. For whatever minimal metadata that needs to be sent on the hare channel, Section 3.1.1 describes our fuzzy header matching technique that is resilient to a moderate number of errors in metadata without using any FEC.

To further avoid eating into the "tortoise" link bandwidth, Section 3.2 describes a multi-resolution CRC technique that provides efficient hierarchical block-level CRCs. We then describe HTPPR 's joint PPR/autorate mechanism, in which the PPR metadata is exploited for improved transmission bit rate selection.

3.1 PPR Mechanism

We select a block-retransmission mechanism for use in HTPPR because it is more amenable to the tortoise-link bandwidth optimization techniques we discuss in the following section. Additionally, our autorate mechanism (Section 3.3) adjusts the amount of FEC applied to the communication. HTPPR divides each data packet into smaller fixed-size units called "blocks". The sender transmits the packet and its sequence number on the hare channel, and transmits the CRCs of each block in the packet (and the sequence number as well) over the tortoise channel. To protect the sequence number, HTPPR applies a modest amount of FEC using a 16,9 Reed-Solomon code. The receiver links these two pieces of information together for data recovery.

Each packet transmitted on the hare channel can be received correctly, completely lost, or corrupted. The receiver must identify which occurred and transmit appropriate feedback to the sender. We decide to use the tortoise link for the receiver feedback. The reasons for that are: 1) Loss of feedback information will impair recovery; 2) Using the tortoise link for feedback allows us to offload small packets from the hare link and also to remove receiver channel contention. If the packet is correct (every block passes the CRC check and the sequence number is the next-expected one), the receiver sends an ACK for that sequence number. If the sequence number is higher than expected, the receiver sends a NACK for the missing packet(s), requesting to retransmit in full (packet #3 in the figure). If one or more blocks fails the CRC check, the receiver must send an ARQ (using block CRCs sent as illustrated in step 3 in Figure 2 for the lost blocks (sequence number and a bitmap indicating which blocks were received correctly). Figure 2 illustrates this for the packet with seq# 2 and protocol step 4.

The basic technique above successfully recovers partially corrupted packets, but it risks imposing long delays if the tortoise link latency is higher than that of the hare link—a common scenario. To solve this problem, HTPPR transmits a burst of packets and, in parallel, notifies the receiver of the maximum sequence number, as shown in Figure 2, step 1. The receiver can thus identify lost packets in two ways: 1) By a missing sequence number; and 2) by a timeout for the last sequence number (the maximum sequence number that is expected) making the protocol slightly more tolerant to latency on the tortoise link. A second problem with this basic technique is that if the loss rate is high, it can require a large number of small packet retransmissions on the wireless link (for the lost blocks). To tackle this problem, we perform data transmission on a larger "chunk" basis.

Chunk-based recovery: Multiple blocks constitute a *chunk* which is designed to be much larger than a hare packet. Each chunk has a unique 20-byte identifier. The sender initiates chunk transmission by telling the receiver the ID of the chunk it should receive next. The sender then transmits over the hare link the packets in a chunk, waiting until the packets are ACKed on the tortoise link before sending the next burst. Concurrently, the sender transmits the maximum sequence number via the tortoise link (step 1 in Figure 2). The sender uses a sliding window of chunks to ensure full utilization of the tortoise link.

Once all the responses for the packets of the current burst (for a particular chunk) are received the sender and receiver know exactly what blocks have been successfully transmitted in the chunk so far. They prepare the next packet burst in the increasing order of the blocks that have to be sent (or expected) and the process continues till the chunk is transferred ensuring in-order delivery of blocks of a chunk. HTPPR uses 32KB chunks, a value much larger than the size of a packet, hence, small packets are transmitted only when the last few blocks of the chunk are being transferred. Hence, the channel contention is reduced further.

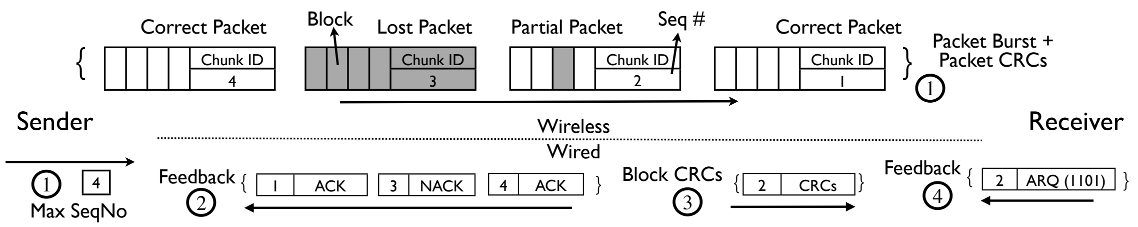


Figure 2: HTPPR Packet Formats and Information Flow Timeline. The greyed out packet contents indicate transmission losses on the wireless link. The circled numbers indicate the steps in the protocol's operation.

3.1.1 Fuzzy matching the chunk identifier

If the chunk identifier is corrupted, the hare packet could be dropped without even inspecting the data. To avoid this pitfall, we use a fuzzy matching strategy to read and match chunk identifiers. The receiver is always notified by the sender about the entire list of chunks being transmitted at every moment. Each incoming packet has to be first categorized on the basis of its chunk identifier. Each incoming hare packet is further processed only when the chunk identifier on the packet "closely" matches an expected chunk identifier. We utilize fuzzy matching as opposed to an FEC based encoding and decoding technique to reduce the overhead from performing FEC over 20 byte chunk identifiers. However, we utilize a (2,1) FEC for encoding and decoding our two byte packet sequence number which increases the overhead only by two bytes.

Our fuzzy matching technique allows for a large number of bits to be erroneously received in a chunk identifier—only 120 of the 160 bits must be correctly received. In the (very unlikely) chance of a false positive in chunk matching[1], the multiresolution CRC technique described in the next section will identify the mismatch. Timeouts along with the maximum sequence number will detect a loss in case of a false negative. Hence, HTPPR harnesses even packets with corrupt headers in most cases.

3.2 CRC at multiple resolutions

The design of HTPPR must ensure that the per-block CRCs sent across the tortoise link do not consume too much bandwidth. The design choices to make are: 1) the size of a block and 2) the size of its CRC. Smaller blocks lead to higher recovery efficiency but also increase the total amount of CRCs. The size of the CRC directly affects the tortoise link traffic. To resolve this tension, HTPPR uses a multi-resolution CRC approach. It provides small (8 bit) CRCs for each small block (32 byte). To ensure that these weak CRCs can still detect errors, the sender computes larger CRCs for groups of blocks: a 32 bit CRC over all blocks in the packet and a 16-bit CRC for the blocks in each quarter of the packet.

[1]The probability is approximately 1 in 35 billion.

For each packet, the sender first transmits the 32 bit CRC over the tortoise link to the receiver. If the packet fails the 32 bit CRC check, the receiver uses 8 bit block CRCs, sent by the sender, to find which blocks are corrupted. In addition, the receiver requests the sender to transmit the four additional 16 bit CRCs to help identify if the 8 bit CRCs failed to detect any errors. If the 16 bit CRC fails on any quarter of the packet for which the 8 bit CRCs did not detect an error, an ARQ for the entire quarter of the packet is transmitted. This scheme incurs a slightly higher delay but significantly reduces the bandwidth requirements on the tortoise links. However, as explained before, a sliding window of chunk transmissions ensures that even if a single chunk transmission is stalled because of the need of additional CRCs, the wireless channel is utilized for other ongoing chunk transmissions.

Now, we have an efficient PPR mechanism for HTLs. But, as noted in ZipTx [17] the real benefits of a PPR scheme are reaped when the sender has the ability to choose a higher wireless transmission rate than non PPR systems and be able to harness the increased number of partial packets.

3.3 Block Error Rate Based Autorate

Systems such as *SoftRate* [23] and ECC [7] have shown that accurate channel bit-error rate estimates can be used for improved wireless bit rate selection. *SoftRate* obtains these estimates using *SoftPHY* hints from a software radio. Our design leverages HTPPR's block-granularity error estimates (much more fine-grained than packet-level loss estimates) as the basis for a *SoftRate*-like bit rate selection mechanism. We term these error estimates the *Block-level error rate*, or BlER. Since blocks are substantially larger ($\approx$32 bytes) than individual bits, a single packet's worth of blocks may not provide enough information for accurate bit-error rate estimation. HTPPR estimates BlER using an exponentially weighted moving average (EWMA) over a small window of recently received packets with a small bias towards recent history. Additionally, in the calculation of BlER, Equation 3 gives less weightage to the packets whose inter-arrival time deviate more from the average inter-arrival time. This helps in a more even sampling of the channel loss rate.

The following set of equations show how we calculate the BlER for a sender-receiver pair. T_i represents the moving average for inter-arrival time of packets, V_i represents the moving standard-deviation for the inter-arrival time of packets, B_i represents the moving average value of block level error estimate with a higher weight on recent block level error updates; all values calculated after the receipt of the i^{th} packet. t_i represents the time at which the i^{th} packet was received and b_i represents the block level error estimate from the i^{th} packet. α and β are protocol parameters.

$$V_i = \beta \cdot (|t_i - t_{i-1} - T_{i-1}|) + (1-\beta) \cdot V_{i-1} \quad (1)$$

$$T_i = \alpha \cdot (t_i - t_{i-1}) + (1-\alpha) \cdot T_{i-1} \quad (2)$$

$$B_i = e^{-\frac{|t_i - t_{i-1} - T_i|}{V_i}} \cdot b_i + (1 - e^{-\frac{|t_i - t_{i-1} - T_i|}{V_i}}) \cdot B_{i-1} \quad (3)$$

Equation 1 and 2 show how we calculate standard deviation and average inter-arrival time for packets as a simple moving average similar to TCP [19]. Equation 3 represents how we give more weight to recent updates in BlER and phase out the older estimation depending on how old it was. Instantaneous BlER is calculated using ACKs and ARQs. Given the BlER at the current bit rate, the sender uses simple heuristics similar to the ones in SoftRate [23] to create an autorate scheme. The heuristic assumes that the BlER, at a given SNR, will increase by a factor of 10 when switching to a higher rate. For each bit rate we fix an optimal BlER range. If the current BlER is in the range then the sender continues with the bit rate otherwise it picks a rate to move towards the optimal rate. For example if the current bit rate is 12Mbps and BlER estimate is more than $\frac{1}{4}$ (i.e. the goodput is less than 9Mbps) then it switches to 9Mbps and sees if that yields a better goodput since the loss rate will likely fall below $\frac{1}{40}$. Similarly, if the current bit rate is 12Mbps and the BlER estimate is less than $\frac{1}{30}$ then it switches to 18Mbps since the error could only jump by a factor of 10 and there is a possibility of getting more than 12Mbps goodput in making the switch.

Such a rate adaptation is performed on a per packet-burst basis. At the end of each packet-burst the bit rate is adapted such that the resulting bandwidth (estimated by using the simple heuristic described above to determine changes in BlER when the bit rate is changed) is the highest that can be obtained under the current channel conditions. In our implementation, we used $\alpha = \frac{1}{8}$ and $\beta = \frac{1}{4}$ (used widely in TCP implementations for RTT estimation). For our purposes, spreading error rate over 4 and 8 packets provides sufficiently fine-grained error estimates for bit-rate adaptation. A 1000-byte packet contains 31 blocks, so 4 packets worth allows a 1%-granularity estimate of BlER.

4 HTPPR Implementation

To understand the practical gains achievable using HTPPR, we implemented a data transfer system on top of it. The system provides reliable and efficient bulk data delivery across the “hare” link, leveraging HTPPR’s metadata to also provide content-based redundancy elimination of both retransmitted and overheard data, akin to the “Ditto” system [8].

The protocol is implemented as a user space daemon, with no hardware requirements beyond an 802.11 radio and no kernel changes. HTPPR comprises approximately 11,000 lines of C++, the bulk of which are for HTPPR-specific data structures. The daemon is implemented as an event-driven system that interfaces with both the hare and tortoise networks, and indexes the local disk that stores chunks. We use a modified version of the FAWN-DS [5] log-structured key-value store to index the chunk and CRC data. Our implementation supports 802.11 hare links and can use any tortoise link.

We place the wireless interface (hare link) in Monitor mode using the Madwifi [18] drivers on Atheros cards and configure the device to deliver packets that have CRC and PHY errors to the kernel. Packets are received and transmitted at the user level using raw sockets via Radiotap [20]. In the current implementation, we assume that every arriving packet potentially belongs to an ongoing chunk transfer.[2] Nodes wishing to exchange data connect to each other using a persistent TCP connection over the tortoise link. We encode the tortoise link messages using Protocol Buffers [9].

Parameter Selection: As discussed earlier, larger chunks reduce the number of small packets, thereby reducing the channel contention. We use 32KB chunks and send 5 packets per burst in the transfer mechanism. These settings reduce the number of small packets to less than 4% in practice. Block size determines the benefits of PPR. A large block size reduces the benefits of PPR while very small block size causes high tortoise link overhead because of the increase in the number of CRCs that are transferred. In practice, we found a 32 byte block size to work well under many settings. However, in the future, we would like to explore a design where the block size would also be adapted to the channel conditions.

4.1 Example Application

To demonstrate how a mechanism like HTPPR would be useful we designed a content-based data sharing application to run atop it. The application targets one HTL scenario: Transferring static multimedia content between homes in a neighborhood where they can reach

[2]In future, this limitation could be relaxed by reinjecting non-HTPPR packets into the kernel for subsequent processing.

each other using 802.11 (at highly variable quality) and also via their reliable, but comparatively slow, broadband uplinks (a scenario that could be driven by ISPs themselves and not necessarily in a p2p fashion). We selected this scenario for our evaluation over, e.g., developing world scenarios or wireless in datacenter, because we could more easily develop a realistic model of the connectivity; our results easily generalize for other HTL settings. Additionally, the choice of multimedia content as the data to be transferred enables the system to tolerate the difference in the latencies of the hare and the tortoise links.

Connectivity scenario: Past studies have shown that 802.11 APs are densely deployed in residential areas [4], and wireless signals often reach several neighbors. In [11], the authors used NetStumbler to perform a measurement study using several homes around Boston, Pittsburgh and Portland. The results indicate that more than 85% of the homes can detect at least two APs with an SNR of 15 or higher and 50% of the homes can detect five such APs.

Although the fanout of each home is reasonably large, the SNRs are low (15–18dB). Hence, HTPPR is well-positioned to increase transfer throughput between homes. Such a capability could be used, e.g., for inter-home cache sharing/redundancy elimination, or collaborative data distribution, backup, and so on. It may also apply to multi-hop wireless networks with multiple wired uplinks. For simplicity, we focus on the caching scenario, with an emphasis on static video content. This content has become a significant fraction of network traffic [3] and can be pre-staged into set top boxes by ISPs. An inter-home cache sharing system could help alleviate pressure on access links. In examining this particular use of HTPPR, we were able to exploit several further optimizations. For example, with static content, we pre-compute block CRCs and chunk hashes and store them redundantly around the network to load balance broadband link demands.

5 HTPPR Evaluation

In this section, we first evaluate the effectiveness of each of the HTPPR-based transport protocol's four mechanisms, alone and in combination: 1) pushing all control traffic onto the tortoise link; 2) using block CRCs to recover information in partial packets; 3) using the block-level error rate information as input to a bit rate selection scheme; and 4) using a content-based framework to pre-stage error detection information for static content. We then evaluate the system end-to-end in a testbed and compare it to alternative solutions.

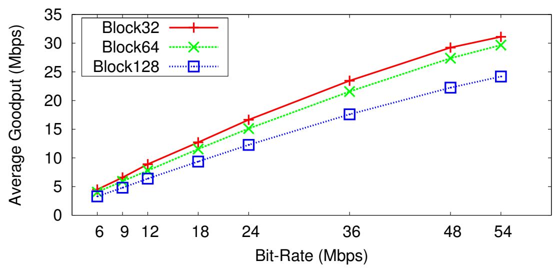


Figure 3: Goodput Vs bit-rate at various block sizes (Five *Block* errors per packet)

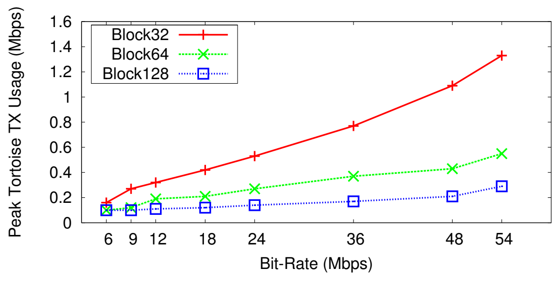
5.1 Control traffic on the tortoise

We start by quantifying the performance improvement provided by offloading control information onto the tortoise link and by having faster NACK generation. This mechanism necessarily offers a minimum performance improvement for HTPPR when compared to a normal wireless UDP or TCP transmission. To isolate the effect of this mechanism, we observe the system in the absence of bit errors (i.e., in regimes of very good SNRs). We placed two hosts close to each other in our lab and configured them to use an unused 802.11g channel, with high antenna power. To understand the requirements for the tortoise link, we leave it effectively unconstrained (a 100 Mbps Ethernet link with 6ms delay) and report the actual utilization. Such an evaluation setting would demonstrate the baseline benefits of using a reliable link to communicate wireless channel information.

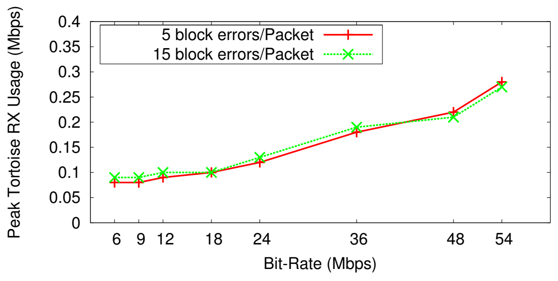
Offloading the control traffic, eliminating channel contention from the receiver and having faster NACK generation for lost packets improved throughput by 11%. HTPPR's goodput was 37.5 Mbps, compared to 33.33 Mbps using UDP iperf. To provide this gain, the hosts required moderate amounts of tortoise link bandwidth: The sender transmitted 0.95 Mbps of metadata (CRCs for partial packets, maximum sequence numbers and chunk IDs) and received 0.21 Mbps of feedback (ACKs, ARQs and NACKs). This experiment goes to show how timely and reliable delivery of minimal channel information can improve wireless bandwidth even at high channel SNRs. We expect the benefits to be larger than 11% when the channel is lossy as we demonstrate in Section 5.4.

5.2 Recovering partial packets

To assess the potential benefits and overheads associated with a sub-packet CRC mechanism, we evaluate the performance of HTPPR under different block sizes, and artificially induced bit errors. We do so using the setting described in the previous section, which had a baseline wireless throughput (UDP iperf) of 33.33Mbps using 802.11g. These microbenchmarks evaluate three different block sizes: 32 bytes, 64 bytes and 128 bytes; each packet can fit 44, 22, and 11 blocks, respectively,



(a) Peak sender tortoise TX usage (15 block errors/Packet)



(b) Peak sender tortoise RX usage (32byte blocks)

Figure 4: Tortoise link TX utilization at various bitrates and Block sizes

for these block sizes after taking the header overhead into account (*Chunk* name and sequence number).

We induce bit errors artificially. We perform three experiments where we purposely corrupt 5 and 15 blocks per packet, introducing at least one bit error in each of these blocks. We select these parameters to stress HTPPR in extreme conditions. Earlier experimental measurements report 40% partial packet error rate, with a 5% byte error rate [17]. In this experiment, we corrupt every packet and with a large number of errors. Such an extreme experimental setup would help in understanding the benefits of a low overhead partial packet recovery mechanism – with only minimal information exchange on the wired link, even extremely unreliable wireless channel can be used for communication.

Our results, presented in Figures 3 and 4, report average goodput and peak tortoise bandwidth use across 20 runs. Each run reflects the performance of a 100MB transfer (CRCs calculated and transmitted on-demand). Figure 3 shows, as expected, at a given block error rate, the goodput is the greatest for block size of 32 bytes. It must be noted that at a given channel condition the block error rate would increase with the size of the block. Hence, it is desirable to have a block size that is as small as possible to increase the efficiency of partial packet recovery. However, too small a block size can increase the overhead from block level CRCs. Our multi-resolution CRC technique allows us to use a block size of as small as 32 bytes with only a $\frac{1}{32}$ factor overhead from block-level CRCs.

The traffic and channel conditions of the hare link determine how much the tortoise link is used for control and feedback messages. Sender tortoise link RX utilization depends on the number of ACKs and retransmit requests (for erroneously received blocks and lost packets). The sender tortoise link TX utilization depends on how much metadata and CRCs are sent (metadata depends on the data rate and the CRCs depend on the number of partial packets). The peak sender tortoise RX utilization is shown as a function of the bit error rate (dependent only on the number of ACKs, ARQs and NACKs). Figure 4 shows the *peak* tortoise link usage.

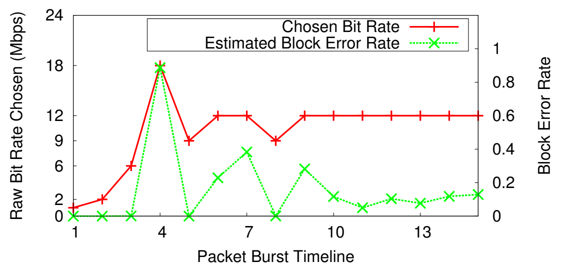


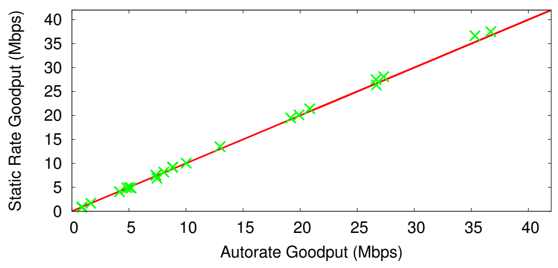
Figure 5: HTPPR Adapts Bit Rate According to BlER

We notice that the peak sender tortoise link TX utilization increases with decreasing block size (Figure 4(a)). That result is expected since now a small number of errors will result in a larger number of CRCs being sent to the receiver. The higher the transmission rate we try to sustain on the hare medium, the more bandwidth we need to consume, since higher wireless bit rates are accompanied by decreased robustness to bit errors. On the receiver side, we observe tortoise link TX utilization also increases with wireless bit rate (Figure 4(b)). Moreover, the more errors the receiver encounters, the more requests for recovery information it needs to transmit to the sender. Since the request for recovery is a bitmap, it makes the sender tortoise link RX utilization insensitive to the *block* error rate.

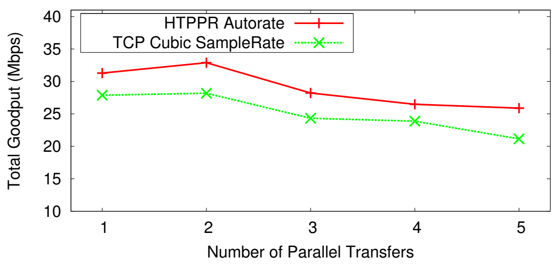
Even under the most aggressive conditions, however, such as a large number of errors and a high wireless bit rate, we observe that the required tortoise link TX utilization for the sender and the receiver do not exceed 1.4Mbps and 0.7Mbps, respectively. Recall that in practice, only up to 40% packets are partial [17], thus the tortoise overhead in real workloads will be much smaller.

5.3 Autorate Performance

The previous section showed how HTPPR can harness partial packets and provide better goodput in when the loss remains constant. In this section, we demonstrate how the autorate scheme described in Section 3.3 helps HTPPR achieve similar improvements in real varying channel conditions. Figure 5 provides a timeline based graph showing how the bit rate selection mechanism



(a) Static Bit Rate vs HTPPR Autorate Over Static Channels



(b) HTPPR Autorate and Multiple Transfers

Figure 6: HTPPR Autorate Performance

adapts to block error rate on a link. Within 7 to 8 packet bursts the system is able to zero in on the ideal bitrate needed for that channel at the start of the experiment.

We evaluate our bit rate selection mechanism in two ways. First, we determine the accuracy of bit rate selection mechanism by benchmarking the protocol on relatively non-varying channels. If the autorate mechanism can deliver the goodput that can be obtained by manually choosing an bit rate that will provide the highest goodput in this channel then the autorate can be considered accurate. Second, we evaluate the bit rate selection scheme by studying its behavior in the presence of multiple data transfers. We create data transfers between some pairs of nodes such that the receivers can hear all the senders. We monitor the aggregate goodput obtained by all the receivers in each case and see how the bit rate selection scheme fares in presence of interference.

Figure 6(a) compares the goodput achieved by HTPPR when using the bit rate selection scheme and when using a static bit rate that obtains the highest goodput in the channel (the red line is the reference). We sampled 25 different links, relatively non-varying, with a range of SNRs, from our experimental setup and transferred 100 MB of data per transfer. The results indicate that the goodput achieved by using the bit rate selection scheme reaches close to the goodput achieved by using a static bit rate which would indeed be the ideal goodput possible in these links. This demonstrates that in channels which are relatively stable our autorate selection scheme can be very effective.

Figure 6(b) shows the collective goodput achieved by all the transfers when there are multiple senders and receivers. As a comparison we show how TCP Cubic with SampleRate works when increasing the number of transfers under similar conditions. HTPPR not only has higher throughput in all cases but also works better when there are multiple transfers going on. As mentioned earlier in Section 2 our current focus is on unidirectional bulk transfers of static multimedia content and we believe that our bit rate adaptation scheme is adequate for such settings.

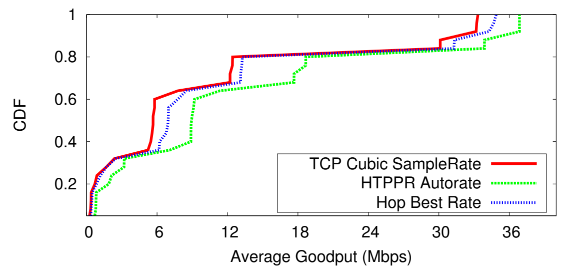
5.4 End to End Performance

The previous results establish the potential for HTPPR to increase goodput in lossy conditions while using only small amounts of tortoise link capacity. In this section, we study the end to end performance of the system and compare it to data transfer protocols using only the hare link, including TCP Cubic with SampleRate, Hop [16] and a block-retransmission based ZipTx. Hop is a wireless transport mechanism that is designed to reduce ACK traffic to a minimum. Its acknowledgment policy reduces the channel contention from the receiver and helps in the increase of bandwidth. Although its design focuses primarily on multihop flows, many of its design decisions lead to improved single-hop performance as well. We compare HTPPR to single-hop Hop flows and also a block-retransmission based ZipTx.

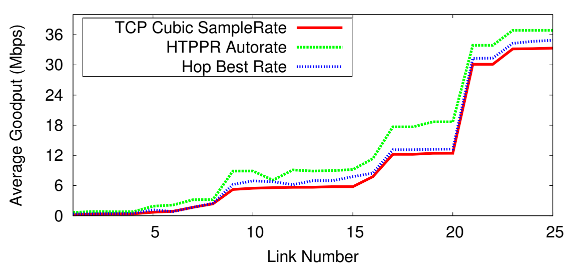
We compare HTPPR using our bit rate selection scheme to TCP Cubic with SampleRate, and to Hop [16] in an 802.11 Adhoc setting. Hop does not have an autorate scheme. As a close approximation, we evaluate Hop at four different bit-rates on each link: 802.11b 11Mbps, 802.11g 24Mbps, 802.11g 36Mbps and 802.11g 54Mbps, and report the best of the four resulting throughput values (Hop BestRate).

To understand any tradeoffs as a function of hare link quality, we perform experiments using twenty five different links in our experimental setup with a range of SNRs from 10–22dB. The performed experiment includes two nodes that wish to transfer a file of 100 MB (unidirectional). We use an average 10ms delay (20 ms ping) on the tortoise link (again 100Mbps Ethernet, although we never use more than 1.5Mbps in either direction on the link in any of our experiments). TCP is benchmarked using iperf. Our results are presented in Figure 7.

Figure 7(a) shows the cumulative distribution function (CDF) of goodput achieved by each of the above mentioned techniques in the 25 links that we benchmarked (note that links with low rates have a low SNR). HTPPR outperforms TCP and Hop over the entire range of link conditions. Hop BestRate outperforms TCP over a range of link settings. More evidently, in Figure 7(b)



(a) HTPPR vs TCP vs Hop



(b) Sorted TCP Goodput

Figure 7: HTPPR vs TCP vs Hop

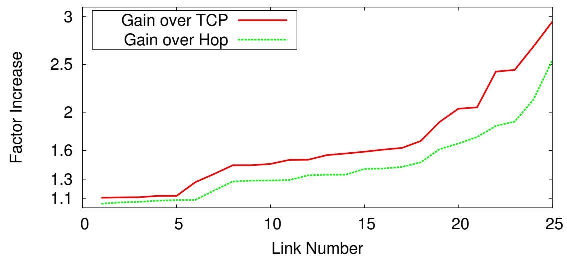


Figure 8: HTPPR 's Gain (factor) over TCP and Hop

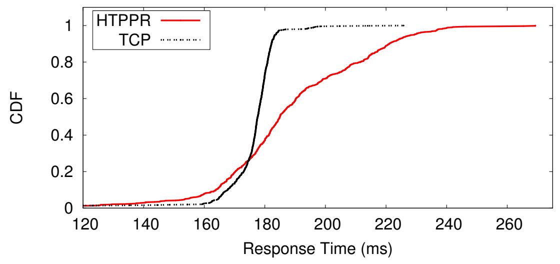


Figure 9: CDF of *Chunk* delays

we report the absolute goodput values obtained over the entire range of links and we report the sorted factor gains obtained over TCP and Hop BestRate in Figure 8. HTPPR performs upto 3x times better than TCP and upto 2.5x times better than Hop. HTPPR improves upon TCP by at least 3x for 10% of links and it improves by at least 11% for all the links. 3x improvement is achieved over links with very poor quality. For example, on one link where TCP obtained a throughput of 223 Kbps, HTPPR obtained a throughput of 658 Kbps.[3] Also, on links with a HTPPR throughput between 2–20Mbps, the gain over TCP is between 100% to 30%.

We also conducted a few experiments with a block based implementation of ZipTx. Instead of sending the metadata over the tortoise link, we send the metadata over the hare link with all-or-nothing packet level CRC protection. This gives insight in to the performance gains that can be obtained by doing a multi-round partial packet recovery entirely over wireless similar to the way it is done in ZipTx. HTPPR achieved 20% to 30% more throughput than ZipTx did.

With feedback to trigger new packets being sent over the tortoise link and with a scheme where CRCs are explicitly asked for, HTPPR can incur slightly higher latency than TCP. Figure 9 plots the cumulative distribution functions (CDF) for *chunk* (32KB) transfer latencies for both protocols at SNR=16dB. As expected, the higher latency of the tortoise link leads to a *Chunk* latency distribution with a higher average value as well as variance, but still less than a factor of 1.5. However, this slight increase of delay is accompanied by a 50% throughput improvement over TCP on such a link.

5.5 Benefits in neighborhood settings

One of our earlier assertions is that HTPPR is an attractive data transfer mechanism for content delivery in neighborhoods using collaborative caching. To test this conjecture, we connect between 2 and 6 nodes to each other via broadband-like links (7.1 Mbps downlink, 750 Kbps uplink). At the terminus of the broadband links, we place a high-speed server node that can act as a point-of-presence CDN mirror inside the ISP.

Impact on the user: In this microbenchmark, we estimate how much time is needed to download two 100 MB files, both needed by two neighbors, in various settings: BBCDN, BBPeer, and HTPPR. BBCDN is representative of what happens today. Both neighbors that want these files fetch them from a CDN. BBPeer represents a peer-to-peer download application like BitTorrent [1]. Here we make each node download one file from the CDN and the other from the neighbor over the broadband using BitTorrent. HTPPR is our system which will fetch the content from the neighborhood when possible via HTPPR or fetch it from the CDN. In each setting we make all the downloads parallel to correctly reflect the ideal gains from each setting. For HTPPR we assume a channel of 15 dB (true more than 85% of the time [11] enabling a channel between the

[3]Why not just send the data over the tortoise?: HTPPR requires only 30Kbps of tortoise bandwidth to gain over 400Kbps of hare-link bandwidth; a more conventional link-striping scheme could be used to consume the rest of the tortoise bandwidth if desired.

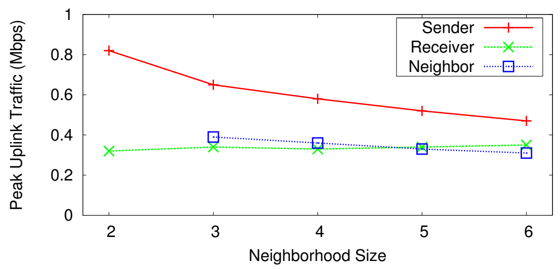


Figure 10: Uplink Aggregation Benefits

neighbors with HTPPR throughput of 9.1 Mbps). The completion times are 178, 450 and 1066 seconds for HTPPR, BBCDN and BBPeer; the total broadband traffic was 227, 412 and 638MB respectively. On this link, using wireless-only TCP would have provided 28.7% less bandwidth when compared to HTPPR, demonstrating its benefits.

Impact on the network: The sender uplink bandwidth required by HTPPR may exceed 1 Mbps and thus be considered prohibitive in some cases. In this section, we evaluate one possible optimization aiming to reduce the uplink bandwidth required on the sender. Using a real deployment of six wireless nodes, also connected to each other via a tortoise link, we show how peak broadband uplink utilization can be reduced when pre computed CRC information is stored uniformly across the neighborhood. Such an optimization is extremely useful when dealing with static multimedia content that has to be shared between homes of a neighborhood – the utilization of the broadband link can be minimized during day time and the CRCs can be precomputed and distributed across the neighborhood during the nighttime. We present the uplink usage statistics at the highest data transfer rate using 802.11g, when the neighborhood size increases from two to six. The assumption here is that the pre computed CRC information is stored uniformly across the neighborhood, during nights and low network usage times for popular static content. We benchmark the wireless transfers between a pair of homes when everyone else in the neighborhood is sending CRC to the receiver along with the sender.

Figure 10 shows how the peak uplink utilization varies with the number of neighbors. As the number of neighbors increases, there is a steady decrease in the uplink utilization of the sender and neighbors. Receiver uplink utilization remains almost the same at all sizes of the neighborhood since the amount of feedback does not decrease with the number of neighbors. With just one additional neighbor the uplink utilization becomes uniformly small across the neighborhood.

6 Related Work

Partial Packet Recovery schemes fall into two broad categories: Those that use physical-layer confidence estimates and those that augment the data with additional metadata to perform error detection and correction.

SOFT [24] operates by accessing the radio's physical layer confidence in decoding each bit. It is a multi-AP scheme targeted at an enterprise setting in which multiple access points hear the same transmission from a client. These APs use their high-bandwidth wired link to share their confidence estimates. The receiving AP can combine confidence estimates from other APs and obtain a correct packet. Similarly, multiple faulty retransmissions can be combined to obtain a correct packet.

In PPR, Jamieson et al. [13] incorporate two new ideas: (1) SoftPHY, an expanded physical layer (PHY) interface that provides PHY-independent hints to higher layers about the confidence in decoding each bit, and (2) a postamble scheme to recover data even when a packet preamble is corrupted and not decodable at the receiver. The resulting asynchronous link-layer ARQ (Automatic Repeat reQuest) protocol, based on PPR, allows a receiver to compactly encode a request for retransmission of only those bits in a packet that are likely in error.

To overcome the limitation of needing physical layer changes (and also using fixed modulation and coding schemes), earlier schemes such as ZipTx [17] and our HTPPR protocol in this paper perform partial packet recovery purely in software. To do so, they and we use a block-based or FEC-based approach. ZipTx operates entirely over wireless (unlike SOFT, it does not require a high-speed wired link between receiving APs). ZipTx uses multi-round data and feedback via NACKs and ARQs to detect and harness partial packets.

Maranello [10] is a partial packet recovery scheme that can be incrementally deployed in existing 802.11 settings. It does not add additional metadata for correct packets, but instead only generates NACKs for corrupted blocks. HTPPR, like Maranello, does not have any additional overhead for correct packets. Additionally, HTPPR has two distinct advantages over Maranello: 1) It does not send any feedback on the wireless channel, virtually eliminating receiver channel contention and 2) It does not incur overhead from small packets since, HTPPR performs partial packet recovery over large chunks and drastically reduces small packets.

All of the schemes above are sensitive to errors in control messages, metadata, and packet headers (protocol headers, CRCs, etc.) [13]. Receivers usually identify context using the packet metadata; errors in the metadata and headers can lead to the rejection of entire packets. HTPPR is far less susceptible to this problem.

Bit-rate selection. While existing bit rate selection mechanisms [6, 12, 25] can be easily be used on the wireless "hare" link of an HTL, work by Vutukuru et al. and Chen et al. demonstrates that more accurate bit rate selection based on channel error rate (SoftRate [23]) can be performed with access to accurate link error rate information (via SoftPHY hints from a software-defined radio). HTPPR's block-level error rate is used as input to the SoftRate heuristics for rate adaptation. Efficient error estimation codes can be designed [7] within data packets, such that fine grained bit error information can be transmitted back to the sender for accurate bit rate adaptation. HTPPR not only performs efficient error estimation but also uses that information for a multi-resolution partial packet recovery unlike [7].

Link bonding. While HTPPR is well-suited to the highly uneven link capabilities in an HTL scenario, existing work on Link selection, flow-level load balancing, and packet-splitting are more viable in a comparable capability setting. Earlier work by Snoeren used multi-link PPP to multiplex on multiple low-speed links [21]. Here, a single flow can span multiple links, at the cost of fragmentation and re-ordering of frames.

7 Conclusion

A reliable data channel between two nodes can be used to increase throughput over a potentially high-bandwidth error-prone wireless channel. HTPPR does so by giving priority to metadata and feedback by sending them over wired channel. HTPPR performs partial packet recovery on the wireless link efficiently by adopting a multi-resolution CRC mechanism that enables efficient use of the wired-link. This provides up to three times improvement over standard TCP and up to 30% improvement over existing PPR mechanisms. The design and implementation of HTPPR also reveal additional benefits when data can be managed using a content-based transfer mechanism. We implemented HTPPR in user space using commodity hardware, eliminating any kernel/driver dependencies or the need for software radios. Our evaluation of the system suggests that HTPPR is an appropriate technique for optimizing throughput in HTL scenarios.

References

[1] BitTorrent. `http://www.bittorrent.com/`, .

[2] OLPC, . `http://wiki.laptop.org/go/Internet`.

[3] Cisco visual networking index: Forecast and methodology, 20082013. `http://www.cisco.com/`, 2009.

[4] A. Akella, G. Judd, S. Seshan, and P. Steenkiste. Self-management in chaotic wireless deployments. In *Proc. ACM Mobicom*, Cologne, Germany, Sept. 2005.

[5] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. FAWN: A fast array of wimpy nodes. In *Proc. 22nd ACM Symposium on Operating Systems Principles (SOSP)*, Big Sky, MT, Oct. 2009.

[6] J. Bicket. Bit-rate selection in wireless networks. Master's thesis, Massachusetts Institute of Technology, Feb. 2005.

[7] B. Chen, Z. Zhou, Y. Zhao, and H. Yu. Efficient error estimating coding: Feasibility and applications. In *Proc. ACM SIGCOMM*, New Delhi, India, Aug. 2010.

[8] F. Dogar, A. Phanishayee, H. Pucha, O. Ruwase, and D. Andersen. Ditto - a system for opportunistic caching in multi-hop wireless mesh networks. In *Proc. ACM MobiCom*, San Francisco, CA, Sept. 2008.

[9] GoogleProtocolBuffers. Protocol Buffers. `http://code.google.com/apis/protocolbuffers`.

[10] B. Han, A. Schulman, F. Gringoli, N. Spring, B. Bhattacharjee, L. Nava, L. Ji, S. Lee, and R. Miller. Maranello: practical partial packet recovery for 802.11. In *Proc. 7th USENIX NSDI*, San Jose, CA, Apr. 2010.

[11] D. Han, A. Agarwala, D. G. Andersen, M. Kaminsky, K. Papagiannaki, and S. Seshan. Mark-and-Sweep: Getting the "inside" scoop on neighborhood networks. In *Proc. Internet Measurement Conference*, Vouliagmeni, Greece, Oct. 2008.

[12] G. Holand, N. Vaidya, and P. Bahl. A rate-adaptive MAC protocol for multihop wirelss networks. In *Proc. ACM Mobicom*, Rome, Italy, July 2001.

[13] K. Jamieson and H. Balakrishnan. PPR: partial packet recovery for wireless networks. In *Proc. ACM SIGCOMM*, Kyoto, Japan, Aug. 2007.

[14] S. Kandula, K. C.-J. Lin, T. Badirkhanli, and D. Katabi. FatVAP: Aggregating AP backhaul capacity to maximize throughput. In *Proc. 5th USENIX NSDI*, San Francisco, CA, Apr. 2008.

[15] S. Kandula, J. Padhye, and P. Bahl. Flyways to de-congest data center networks. In *Proc. ACM Hotnets-VIII*, New York City, NY. USA., Oct. 2009.

[16] M. Li, D. Agrawal, D. Ganesan, and A. Venkataramani. Block-switched networks: a new paradigm for wireless transport. In *Proc. 6th USENIX NSDI*, Boston, MA, Apr. 2009.

[17] K. C.-J. Lin, N. Kushman, and D. Katabi. ZipTx: Harnessing partial packets in 802.11 networks. In *Proc. ACM MobiCom*, San Francisco, CA, Sept. 2008.

[18] Madwifi. Madwifi. `http://madwifi.org`.

[19] V. Paxson and M. Allman. *Computing TCP's Retransmission Timer*. Internet Engineering Task Force, Nov. 2000. RFC 2988.

[20] Radiotap. Radiotap. `http://radiotap.org`.

[21] A. C. Snoeren. Adaptive inverse multiplexing for wide-area wireless networks. In *Proc. IEEE Conference on Global Communications (GlobeCom)*, pages 1665–1672, Rio de Janiero, Brazil, Dec. 1999.

[22] M. Stemm and R. H. Katz. Vertical handoffs in wireless overlay networks. *Mobile Networks and Applications*, 3(4):335–350, 1998.

[23] M. Vutukuru, H. Balakrishnan, and K. Jamieson. Cross-layer wireless bit rate adaptation. In *Proc. ACM SIGCOMM*, Barcelona, Spain, Aug. 2009.

[24] G. R. Woo, P. Kheradpour, D. Shen, and D. Katabi. Beyond the bits: cooperative packet recovery using physical layer information. In *Proc. ACM Mobicom*, Montreal, Canada, Sept. 2007.

[25] J. Zhang, K. Tan, J. Zhao, H. Wu, and Y. Zhang. Robust rate adaptation for 802.11 wireless networks. In *Proc. IEEE INFOCOM*, Phoenix, AZ, Apr. 2008.

[26] X. Zhao, C. Castelluccia, and M. Baker. Flexible network support for mobile hosts. In *ACM MONET*, pages 137–149, 2001.