Practical Partial Packet Recovery for 802.11: Maranello

COS 598a: Wireless Networking and Sensing Systems

Kyle Jamieson
Context: Coping with wireless bit errors

- **Wired links** are usually all-or-nothing
  - Either the packet arrives correctly or the link is "cut"

- **Wireless links** often deliver packets with errors
  - Bit error rate depends on interference from other links, and multipath fading (recall Roofnet experiments)
  - Packets may have only a few, localized errors

  - Or, packets may have mostly errored bits, but a small piece of salvageable content
Idea: Partial Packet Recovery

• When a frame is received with bit errors:
  – Sender retransmits just the bits that need correcting
  – Receiver combines original transmission with retransmissions to form a correct packet

• Increases throughput, because:
  1. The retransmission is smaller than the original
  2. Shorter transmissions have higher delivery probability than longer transmissions
  3. Consequently, senders select higher bit rates
Alternate approach #1: Block checksum

- Divide each packet into blocks
  - Each block has a one-byte sequence number, 8-bit CRC

- Receiver requests retransmit of just blocks that fail checksum by replying with a **negative acknowledgement (NACK) frame**
  - NACK frame specifies incorrect block sequence numbers

- **Pay block checksum overhead** in the common case (no errors)

| corrupt packet | ✓ | ✓ | ✖ | ✓ | ✖ | Block checksums |
Alternate approach #2: Forward Error Correction

• Don’t attempt to identify correct/incorrect bits
  – Instead send parity bits that contain information about every bit in the packet
  – Common parity coding scheme: Reed-Solomon (R-S)

• Example: ZipTx [MobiCom ‘08]
  – Two-round forward error correction mechanism
    • In 1\textsuperscript{st} round, transmitter sends a small number of R-S parity bits for a corrupted packet
    • In 2\textsuperscript{nd} round, transmitter sends more R-S parity bits
  – If both rounds fail, the receiver requests a retransmission of the whole packet
Alternate approach #3: Physical-layer hints

- Physical layer “scores” each bit with a numerical confidence in that bit’s correctness, passes score up to higher layers.

- Receiver’s link layer asks for retransmissions of just the bits from low-confidence part of the frame.

- Many different ways of combining retransmissions with original transmission.
  - Example: SOFT [MobiCom ‘07] combining information from multiple access points that receive a frame.

![Diagram of Link layer and Physical layer with confidence and received frame.](image)
Maranello

- Block-based checksum design implemented on commodity 802.11 hardware (Broadcom)

- Novel overhead-free link-layer design for the case of no wireless bit errors (common case)

- Maranello protocol implemented in firmware (software running on a small microprocessor on the Broadcom 802.11 network interface card)
Receiver computes link-layer frame checksum, compares to the 802.11 frame checksum field, begins recovery if they don’t match:

1. **Receiver** breaks errored frame into fixed-size *blocks*
   - Sender and receiver agree on the block size beforehand
2. **Receiver** computes Fletcher-32 block checksums for each block and includes all block checksums in a NACK reply
   - If NACK reply lost, transmitter resends entire packet

3. Sender computes block checksums over each block
   - Compares computed block checksums to received block checksums to determine errored blocks
4. Sender transmits *repair blocks* corresponding to just the blocks received that contain errors (*repair packet*)
   - Sender doubles contention window before repair transmission (recall bounded exponential backoff)
   - If sender’s ACK timer expires, it retransmits repair packet
   - If repair packet contains errors, receiver transmits nothing
     - Sender then retransmits the original frame
5. Receiver repairs original transmission with repair blocks
   - Re-computes and verifies a CRC-32 *frame checksum* (computed over entire frame) to check that the recovered packet is indeed correct
Interoperation with legacy 802.11

- **802.11 sender** with Maranello receiver
  - Does not recognize Maranello NACK from receiver
  - So sender retransmits as normal after the (short) “ack timeout” period

- Maranello sender with **802.11 receiver**
  - Just sends 802.11 ACK if correct, nothing if incorrect
  - Maranello sender will retransmit entire frame to the 802.11 receiver after ack timeout
Errored bit location by packet

- Orange dot indicates a bit error
  - In packet corresponding to vertical axis position
  - At location in that packet corresponding to horizontal axis position
- Some packets have few bit errors (hypothesis: noise burst?)
  - Errors are mostly restricted to certain 64-byte blocks
  - Can be recovered by retransmitting those blocks
Wi-Fi bit errors cluster together

- Some packets have many bit errors (hypothesis: interference or loss of synchronization)
  - Similarly, can be recovered by retransmitting errored blocks
How many blocks are needed to repair?

- Horizontal axis: number of bit errors in packets
- Vertical axis: Stacked bar graph (# 64-byte blocks required to repair)
One block fixes a one-bit error

- For packets with 1 bit error, one 64-byte block always repairs the packet
One block usually fixes two bit errors

- Among packets with two bit errors:
  - One 64-byte block repairs the packet 99.7% of the time
  - Two 64-byte blocks repair the packet 0.3% of the time
How many bit errors can one block fix?

- Fraction of packets repaired by one 64-byte block, by number of errored bits
  - Under 15 errored bits, \( \approx 90\% \) packets can be fixed with one block
  - Packet size 1,500 bytes, so one block is \( \approx 4\% \) of the packet’s size
How many blocks are needed to repair?

- There are 23 blocks per packet, so orange area represents packets that need a complete retransmission (very few)
Summary: How many repair blocks?

- So far, we have seen the following:
  - The overhead of one block is 4% of a packet
  - For 1–2 errored bits, one block fixes most packets
  - Under 15 errored bits, one block fixes ≈ 90% packets
  - Very few packets require a complete retransmission

- But is number of repair blocks required the right question?
  - We are looking for evidence that partial packet recovery will in fact increase performance, i.e., throughput

  - This data doesn’t tell us anything about how often how many bit errors occur (i.e., where on the x-axis are we most of the time)
Repair size

• Measure how many repair bits (on average) a particular protocol needs to fix one incorrect bit

• Trace-driven simulation
  – Use Broadcom cards to send and receive packets with known payloads over the air
  – Record *traces* of the received frames and mark each received bit as correct or incorrect
  – Software simulator runs the protocol to be evaluated, in simulation, using trace data for received frames
Repair size: Maranello competitive at low BER

- **At low BER:** Maranello requires only marginally more repair bits than Reed-Solomon ZipTx approach

- **Parity bits fix a small number of errors efficiently** (simulated “ideal” ZipTx that knows the number of errors needing repair)
Repair size: Maranello outperforms ZipTx at high BER

- At **high BER**: Maranello outperforms Reed-Solomon based approach
- Additional Reed-Solomon parity bits contain information about the entire packet, **inefficient** if errors are localized to a single block(s)
Retransmission behavior varies across hardware

- **Backoff** and **bit rate selection** impact Maranello’s performance
  - 802.11 standard specifies **backoff** (but chipsets do not always respect the standard)
  - Recall: Standard doesn’t specify bit rate selection

- **Maranello helps Intel** because it increases delivery rate at high bitrates, avoiding backoff

- **Maranello helps Atheros** because it reduces the chance of falling back to 1 Mbit/s
Implementation: Alternatives

• Implementation in the OS kernel driver software
  – Microbenchmark shows > 70 μs (≫ 10 μs SIFS time) delay between receipt of packet and triggered response, so unsuitable
    • CPU interrupt latency and NIC-RAM bus transfer delay

• Software-defined radio platforms (e.g. GNU radio)
  – High-latency (ms) Ethernet or USB bus makes unsuitable

• Sora software defined radio [NSDI ‘08]
  – Software-defined radio on PCI express bus
  – Open question as to whether Sora would work for partial packet recovery (ACKs cached in current version of Sora)
Implementation

- **OpenFWWF** open firmware for Broadcom 802.11 NIC
  - Publically-downloadable firmware assembly code that runs on Broadcom NIC microprocessor

- Broadcom 802.11 NIC system components:
  1. **Tx/Rx FIFO queues**: buffers frames to/from the physical layer (transmission over the air)
  2. **Internal shared memory**: State variables that can be read/written from the kernel driver
  3. **Template RAM**: “Scratch” memory for composing an arbitrary frame and transmitting over the air
  4. **Internal registers and external conditions**: Interface with the physical layer and timers (for, e.g., backoff)
Implementation: NACK generation

• Receiver computes block checksums in firmware

• Problem: For some transmission rates, Maranello NACK airtime is greater than 802.11 ACK
  – May cause problems if hidden terminals present. Why?

  ![Diagram showing H, S, and R with NACK and Data arrows]

• Data contains network allocation vector (NAV) but with a duration shorter than Maranello needs for NACK

• Solution: No solution; just let the collisions happen. Claim that preliminary experiments show improved overall throughput
Implementation: NACK generation

- Problem: 802.11 NIC microprocessor is not fast enough to compute block checksums during SIFS interval (10 μs)
  - Each block checksum takes up to 4 μs
  - But running 802.11, the microprocessor is normally idle during a frame reception

- Solution: Modify firmware to copy partially-received packets into memory the microprocessor can access
  - Overlap one block’s block checksum computation with reception of the next block
Implementation: Sender-side

- Before the first transmission, sender pre-computes block checksums in the OS kernel driver, on the main CPU
  - Then sends block checksums to the firmware with the packet’s contents

- Why? Main CPU is more powerful, can spare the time, and block checksum computation on the sender is not time-critical (why?)
Performance evaluation

- 802.11 channels 1, 6, 11 (span the 2.4 GHz unlicensed frequency band) in environments with background traffic
  - **Advantage:** Characterizes performance of Maranello *in situ*
    - Evaluate in three different environments (research lab, home, university), so can claim some generality

  - **Disadvantage:** Lose repeatability of the experiment, so more difficult for the experimenters to isolate experimental factors that impact performance

- Enable Minstrel **bit rate adaptation**
  - So compare Maranello and 802.11 at or close to the best bit rate for a particular link

- Evaluate throughput, latency gains, and then drill-down for causes
Link throughput experiment

• By how much does Maranello increase link throughput?

• Methodology
  – Use the \textit{Iperf} network measurement tool in UDP mode to saturate a wireless link in the testbed
  – A one-minute run for 802.11, then immediately afterwards, a one-minute run for Maranello
    • < 15 second gap implies wireless conditions \textit{unlikely} to have changed
  – Repeat the experiment ten times with sender and receiver in the same locations
  – Change locations of sender and receiver, in the same testbed
Maranello increases link throughput

- University building results
  - Best results (high channel contention)
  - Other environments qualitatively similar
  - Each point in the scatter plot represents an Iperf run

- Slanted lines delineate constant-factor gains

- Results:
  - About one-third of the time little to no gain
  - About one-third of the time almost 2× gain

Figure 6: Maranello has a higher throughput than 802.11. Each figure compares 802.11 with Maranello in a different environment, or to show the uncertainty of the comparison, with 802.11 itself. Each point represents the performance of back-to-back one-minute UDP throughput measurements; ten points were collected for each configuration of sender and receiver stations.

Figure 7: With block-based repair, Maranello recovers packets faster than 802.11's retransmissions.

6.3 The Sources of Throughput Gain and Latency Reduction
To break down the sources of performance improvement, we enhance the transmission status report for each packet with the following information: (1) whether a repair packet was used, (2) if used, at which attempt, and (3) the number of retransmitted blocks in the repair packet. The original report also includes (1) whether the packet is successfully delivered, (2) the number of attempts, (3) the bit rate used for the packet. With this information, we can calculate the delivery probability at each attempt, the transmission airtime and the number of transmitted bytes for each attempt. We run Iperf for one minute for 10 randomly selected links and plot in Figure 8 the probability of successful attempt for two retransmission rate fallback schemes: Linux “minstrel” fallback which always uses 1 Mbps as fallback rate, and 2-step fallback which drops the bit rate selected by minstrel for the initial transmissions by 2 steps (if possible) and uses it as fallback rate. The two-step fallback selection emulates the Broadcom driver for Windows XP (Section 4.1). In this figure, the x-axis is transmission attempt. The retry limit of Broadcom cards is 7, 1 initial transmission, and at most 6 retransmissions. The y-axis is the probability that an attempt can succeed.
Transmission latency experiment

• Does Maranello decrease the time it takes to correctly deliver one packet across a link?

• Methodology
  – Measure time from the firmware fetching a packet from the head of the Tx FIFO queue, to receipt of an ACK
  – Includes retransmissions (in the case of 802.11), repair transmissions (in the case of Maranello), backoff, etc.
  – Firmware’s microsecond timestamp counter measures this time precisely

  – Desired: Measure time from fetching a packet from the head of the Tx FIFO queue to packet’s correct reception
    • In most cases, this would be a fixed time interval less than proposed measurement (time to deliver the ACK to sender)
Maranello decreases transmission latency

- Latency for packets that need one or more retransmissions
- One pair of sender, receiver locations
- 802.11 modes at 16 and 32 ms represent Minstrel 1 Mbit/s fallback
- A log scale on the x-axis would show more detail at lower latencies — Possibly showing the high-rate retransmissions
Source of Maranello’s improvements

- Measure delivery probability of each transmission attempt
  - Higher delivery probability $\rightarrow$ higher throughput, lower latency

- Note: this graph counts transmissions (including first transmission)

- Attempt #1: Roughly equal between 802.11, Maranello (both just send the original packet)

- Attempts #2, #3: Both 802.11 and Maranello maintain bit rate
  - But, Maranello sends a shorter repair packet
  - Shorter packet has a lesser chance of being lost

- Minstrel fallback to 1 Mbit/s on attempt #4 increases delivery probability
  - Maranello still sending shorter repair packets
Frame aggregation and optimal block size

- 802.11 frame aggregation
  - As bit rates increase, relative overhead of DIFS, backoff, headers, and ACK increases
  - 802.11n, 802.11ac aggregate many frames together
    - Each frame gets own checksum (like “block checksum” approach)
  - Aggregation increases latency
  - Maranello is complementary with aggregation: can repair corrupted aggregates

- Optimal Maranello block size
  - Larger block size would be less efficient on wireless channel but more computationally efficient
  - Can dynamically vary the block size based on BER
Further reading

• ZipTx: Harnessing Partial Packets in 802.11 Networks. Kate Ching-Ju Lin, Nate Kushman, Dina Katabi. Proceedings of MobiCom 2008
  — A practical implementation and evaluation of the forward error correction approach to the problem of partially-received packets


