Practical Partial Packet Recovery for 802.11: Maranello

Brad Karp
(with slides contributed by Kyle Jamieson)
UCL Computer Science

CS M038 / GZ06
3rd February 2016
**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 dependent on interference from other links, fading (recall Roofnet experiments)
  - Packets may have only a few, localized bit-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**
Approach #1: Block Checksum

- Divide each packet into blocks
  - Each block has a one-byte sequence number
  - Each block has an eight-bit CRC checksum

- 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)

- Examples: Seda, FRJ
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 1st round, transmitter sends a small number of R-S parity bits for a corrupted packet
    - In 2nd round, transmitter sends more R-S parity bits
    - If both rounds fail, the receiver requests a retransmission of the whole packet
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.

```
Link layer

Physical layer

Confidence

Received frame: Correct bits Incorrect bits
```
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)
Maranello: Protocol

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 the NACK is 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 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
• Orange dot indicates a bit error
  – In frame corresponding to vertical axis position
  – At location in that frame 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
• 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)
Among all packets with one 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
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

  – Do these data 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 bit-rates, avoiding backoff

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

1. 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

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

3. Sora software defined radio [NSDI ‘08]
   – Software-defined radio on PCI express bus
   – Open question as to whether it 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 low transmission rates, Maranello NACK airtime is greater than 802.11 ACK
  – May cause problems if hidden terminals present. Why?

• 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: Repair Packet Construction

- Transmitter starts an **ACK timer** after sending a frame

- Sender always retransmits the first block of a packet, “which contains important headers of various layers”
  - Why is this necessary?

- Repair packets contain an extra 32-bit checksum in the last four bytes of link-layer data payload
  - Why is this necessary, given 802.11 has link-layer checksum?
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 **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 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.

Figure 8 shows that the probability of successful re-
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

  – Alternative: 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
Maranello Is More Efficient

- Fraction of **effective time**
  - Time spent transmitting correct blocks that have not yet been received

- Maranello *doesn’t throw away correct bits* as 802.11 does
  - Therefore Maranello increases effective time
Deployment on Access Points

• So far, we’ve only seen Maranello in isolation on a link
  – How well does Maranello perform in an AP network with multiple clients?

• Methodology
  – Deploy Maranello on Linksys wireless routers
  – Associate two desktop clients (A, B) with a Maranello AP
  – Send uplink (to the AP) UDP traffic from A and B using Iperf
  – Downlink traffic is more common by volume, but would not show how senders interact
• Both stations running Maranello: significant throughput increase
AP Deployment: Maranello Increases Throughput

- Both stations running Maranello: significant throughput increase
- Running Maranello on one station increases throughput of other
  - No explanation given in paper

Figure 10: With two clients sending to an AP, on average, Maranello increases their individual and overall throughput. Error bars indicate min and max for five one minute runs.

7 Discussion

7.1 Frame Aggregation and Maranello are Complementary
To increase throughput, 802.11n reduces the 802.11 protocol overheads, such as interframe spacing, PHY layer headers and acknowledgment frames, by aggregating data packets into jumbo frames. Aggregated packets that are received incorrectly are indicated in a block acknowledgment which is sent back to the transmitter. The transmitter can then send a new chunk that contains only the corrupt packets. Even though only part of a packet may have errors in it, 802.11n frame aggregation must retransmit whole packets: correctly received bits are wasted.

Frame combining can improve throughput, but it also significantly increases latency, as senders must wait to aggregate enough frames to fill a jumbo frame. Block acknowledgments provide a complementary aggregation of feedback for 802.11n, where ACKs may be buffered together and sent as a group, similarly increasing per-packet latency. Maranello is complementary with these frame aggregation techniques because by repairing corrupted aggregated packets, Maranello can further increase link throughput.

7.2 Optimal Block Size
The Maranello block size is 64 bytes, primarily because it is the smallest multiple of 32 that can be supported by hardware (Section 5.2.2). A larger block size would increase computation efficiency somewhat and shorten NACKs, which may be useful at low bit rates. When the error rate is low, however, larger blocks may lead to repair packets with unnecessary extra bytes, wasting channel time.

We consider an interesting future direction of research to be dynamically adjusting the block size. The ideal block size may vary based on an estimate of wireless channel conditions and the bit rate chosen by the transmitter, which determines the bit rate of the acknowledgments and thus the transmission time of the NACK. When the NACK is transmitted at a low rate, it may be better for global throughput to keep NACK transmissions short than to be precise about the blocks in error. A similar tradeoff exists in the FEC systems between the coding rate of error correction bits and recovery efficiency. Another approach to determine the optimal block size that we intend to explore is to use theoretical models of wireless communication errors [13, 29].

8 Conclusion
In this paper, we design, implement, and evaluate Maranello, a practical partial packet recovery protocol for 802.11 wireless networks. Maranello has the following features simultaneously: (a) it introduces no extra