Administrivia

- Snowcast milestone implementation:
  - Need to include a Makefile, place binaries in top-level directory
  - **Deadline extended to 10pm TONIGHT (Feb 8) so you can do this**
  - To submit, just push your code—we’ll pull the latest commit on `main`
- My office hours: 12-2pm EDT today (group-style, CIT506)
- Snowcast: Due Wed, Feb 16
- HW1: out next class, due after Snowcast
SERVER

CONTROL
127.0.0.1 IPv4
::1 IPv6

LISTENER
BIND
IPv4
Roadmap

• Last time
  – Physical layer: encoding, modulation

• Today
  – Link layer framing
  – Getting frames across: reliability, performance
Layers, Services, Protocols

Physical
- Service: move bits to other node across link

Link
- Service: move frames to other node across link
- May add reliability, medium access control

Network
- Service: move packets to any other node in the network
- IP: Unreliable, best-effort service model

Transport
- Service: multiplexing applications
- Reliable byte stream to other node (TCP), Unreliable datagram (UDP)

Application
- Service: user-facing application
- Application-defined messages
Link Layer
Framing
Framing
Framing

• Given a stream of bits, how can we represent boundaries?
• Break sequence of bits into a frame
• Typically done by network adaptor, or interface
Representing Boundaries

- Sentinels
- Length counts
- Clock-based
Sentinel-based Framing

![Diagram of Sentinel-based Framing]

- **SYN**: 8 bits
- **SYN**: 8 bits
- **HOS**: 8 bits
- **Header**: 8 bits
- **STX**: 8 bits
- **Body**: 16 bits
- **ETX**: 8 bits
- **CRC**: 16 bits
Sentinel-based Framing

- Byte-oriented protocols (e.g. BISYNC, PPP)
  - Place special bytes (SOH, ETX, ...) in the beginning, end of messages
Sentinel-based Framing

- Byte-oriented protocols (e.g. BISYNC, PPP)
  - Place special bytes (SOH, ETX,...) in the beginning, end of messages

- What if ETX appears in the body?
  - Escape ETX byte by prefixing DEL byte
  - Escape DEL byte by prefixing DEL byte
  - Technique known as character stuffing
Sentinels: Bit-Oriented Framing

- View message as a stream of bits, not bytes
- Can use sentinel approach as well (e.g., USB, HDLC)

- HDLC begin/end sequence 01111110
Sentinels: Bit-Oriented Framing

• View message as a stream of bits, not bytes
• Can use sentinel approach as well (e.g., USB, HDLC)
  – HDLC begin/end sequence 01111110
• Use bit stuffing to escape 01111110
  – Always append 0 after five consecutive 1s in data
  – After five 1s, receiver uses next two bits to decide if stuffed, end of frame, or error.
Length-based Framing

- Problem with sentinels: Length of frame only on data seen
- Alternative: put length in header (e.g., DDCMP)
Length-based Framing

• Problem with sentinels: Length of frame only on data seen
• Alternative: put length in header (e.g., DDCMP)

- Danger: Framing Errors
  - What if high bit of counter gets corrupted?
  - Adds 8K to length of frame, may lose many frames
  - CRC checksum helps detect error
Clock-based Framing

E.g., SONET (Synchronous Optical Network)
- Each frame is 125μs long
- Look for header every 125μs
Error Detection

• Basic idea: use a checksum
  – Compute small checksum value, like a hash of packet

• Good checksum algorithms
  – Want several properties, e.g., detect any single-bit error
  – Details later
Error Detection and Correction
Error Detection

• Idea: have some codes be invalid
  – Must add bits to catch errors in packet
Error Detection

• Idea: have some codes be invalid
  – Must add bits to catch errors in packet

• Sometimes can also correct errors
  – If enough redundancy
  – Might have to retransmit
Error Detection

• Idea: have some codes be invalid
  – Must add bits to catch errors in packet

• Sometimes can also correct errors
  – If enough redundancy
  – Might have to retransmit

• Used in multiple layers
Simplest Schemes
Simplest Schemes

- Repeat frame n times
Simplest Schemes

- Repeat frame n times
  - Can we detect errors?
Simplest Schemes

- Repeat frame n times
  - Can we detect errors?
  - Can we correct errors?
Simplest Schemes

- Repeat frame n times
  - Can we detect errors?
  - Can we correct errors?
    - Voting
Simplest Schemes

• Repeat frame n times
  – Can we detect errors?
  – Can we correct errors?
    • Voting
  – Problem: high redundancy : n
Simplest Schemes

• Repeat frame n times
  – Can we detect errors?
  – Can we correct errors?
    • Voting
      – Problem: high redundancy : n
• Example: send each bit 3 times
Simplest Schemes

• Repeat frame n times
  – Can we detect errors?
  – Can we correct errors?
    • Voting
      – Problem: high redundancy : n

• Example: send each bit 3 times
  – Valid codes: 000 111
Simplest Schemes

• Repeat frame n times
  – Can we detect errors?
  – Can we correct errors?
    • Voting
      – Problem: high redundancy : n

• Example: send each bit 3 times
  – Valid codes: 000 111
  – Invalid codes: 001 010 011 100 101 110
Simplest Schemes

- Repeat frame n times
  - Can we detect errors?
  - Can we correct errors?
    - Voting
      - Problem: high redundancy : n

- Example: send each bit 3 times
  - Valid codes: 000 111
  - Invalid codes: 001 010 011 100 101 110
  - Corrections: 0 0 1 0 1 1
Parity

Add a parity bit to the end of a word
Parity

Add a parity bit to the end of a word

• Example with 2 bits:
  – Valid: 000 011 101 110
  – Invalid: 001 010 010 111

  Can we correct?

  No, but can detect
Add a parity bit to the end of a word

- Example with 2 bits:
  - Valid: 000 011 101 110
  - Invalid: 001 010 010 111

- Can we correct?

- Can detect odd number of bit errors
  - No correction
In general
In general

Hamming distance: number of bits that are different between two codes

- E.g.: HD (00001010, 01000110) = 3

01000110

0100110
In general

Hamming distance: number of bits that are different between two codes

- E.g.: HD (00001010, 01000110) = 3

- If min HD between valid codewords is $d$:
  - Can detect $d-1$ bit error
  - Can correct $\lfloor (d-1)/2 \rfloor$ bit errors
In general

Hamming distance: number of bits that are different between two codes

- E.g.: HD \((00001010, 01000110)\) = 3

- If min HD between valid codewords is \(d\):
  - Can detect \(d-1\) bit error
  - Can correct \(\lfloor (d-1)/2 \rfloor\) bit errors

- What is \(d\) for parity and 3-voting?
  \[ d = 2 \quad \Rightarrow \quad d = 3 \]
Getting Frames Across: Reliability and Performance
Sending Frames Across

Transmission Delay

Propagation Delay

Latency

A → B
Sending Frames Across

Throughput: bits / s
Which matters most, bandwidth or delay?
Which matters most, bandwidth or delay?

• How much data can we send during one RTT?
Which matters most, bandwidth or delay?

- How much data can we send during one RTT?
- E.g., send request, receive file
Which matters most, bandwidth or delay?

- How much data can we send during one RTT?
- E.g., send request, receive file

For small transfers, latency more important, for bulk, throughput more important
Performance Metrics

- **Throughput**: Number of bits received/unit of time
  - e.g. 10Mbps
    - \(10 \text{ Mbps} = 10 \text{ MBITS/SEC} = 10 \times 10^6 \text{ BITS/SEC}\)
Performance Metrics

• **Throughput**: Number of bits received/unit of time
  – e.g. 10Mbps

• **Goodput**: Useful bits received per unit of time
Performance Metrics

- **Throughput**: Number of bits received/unit of time
  - e.g. 10Mbps

- **Goodput**: Useful bits received per unit of time

- **Latency**: How long for message to cross network
Performance Metrics

• **Throughput**: Number of bits received/unit of time
  – e.g. 10Mbps

• **Goodput**: Useful bits received per unit of time

• **Latency**: How long for message to cross network

• **Jitter**: Variation in latency
Components of Latency
Components of Latency

• Processing
  – Per message, small, limits throughput
  – e.g. $100 \frac{Mb}{s} \times \frac{pkt}{1500 B} \times \frac{B}{8b} \approx 8,333 \frac{pkt}{s}$

120μs/pkt
Components of Latency

• Processing
  – Per message, small, limits throughput
  – e.g. \( \frac{100 \, Mb}{s} \times \frac{pkt}{1500 \, B} \times \frac{B}{8b} = \frac{8333 \, pkt}{s} \approx 120 \, \mu s/pkt \)
Components of Latency

- **Processing**
  - Per message, small, limits throughput
  - e.g. \[ \frac{100 \text{Mb}}{s} \times \frac{\text{pkt}}{1500 \text{B}} \times \frac{B}{8b} = 8333 \text{ pkt/s} \approx 120\mu s/\text{pkt} \]

- **Queue**
  - Highly variable, offered load vs outgoing b/w

\[ \implies \text{DELAY INSIDE SWITCH/ROUTER} \]
Components of Latency

• **Processing**
  - Per message, small, limits throughput
  - e.g. \[
  \frac{100 \text{ Mb}}{s} \times \frac{\text{pkt}}{1500 \text{ B}} \times \frac{B}{8b} \approx 8333 \text{ pkt/s}
  \]
  \[= 120 \mu s/\text{pkt} \]

• **Queue**
  - Highly variable, offered load vs outgoing b/w

• **Transmission**
  - Size/Bandwidth

TIME TO SEND ON AN INTERFACE
Components of Latency

- **Processing**
  - Per message, small, limits throughput
  - e.g. \[ \frac{100 \, Mb}{s} \times \frac{pkt}{1500 \, B} \times \frac{B}{8 \, b} \approx 8,333 \, pkt/s \] \( \approx 120 \mu s/\text{pkt} \)

- **Queue**
  - Highly variable, offered load vs outgoing b/w

- **Transmission**
  - Size/Bandwidth
Components of Latency

• Processing
  – Per message, small, limits throughput
  – e.g. \( \frac{100 \text{Mb}}{s} \times \frac{\text{pkt}}{1500 \text{B}} \times \frac{\text{B}}{8b} \approx 120 \mu \text{s/pkt} \)

• Queue
  – Highly variable, offered load vs outgoing b/w

• Transmission
  – Size/Bandwidth

• Propagation
  – Distance/Speed of Light
Reliable Delivery
Reliable Delivery

• Several sources of errors in transmission
• Error detection can discard bad frames
• Problem: if bad packets are lost, how can we ensure reliable delivery?
  – Exactly-once semantics = at least once + at most once
At Least Once Semantics
At Least Once Semantics

• How can the sender know packet arrived at least once?
  – Acknowledgments + Timeout

\[ \Rightarrow \text{ACK} \]
At Least Once Semantics

• How can the sender know packet arrived at least once?
  – Acknowledgments + Timeout

• Stop and Wait Protocol
  – S: Send packet, wait
  – R: Receive packet, send ACK
  – S: Receive ACK, send next packet
  – S: No ACK, timeout and retransmit
Stop and Wait Problems

• Duplicate data
• Duplicate acks
• Slow (channel idle most of the time!)
• May be difficult to set the timeout value
Duplicate data: adding sequence numbers
At Most Once Semantics

• How to avoid duplicates?
  – Uniquely identify each packet
  – Have receiver and sender remember

• Stop and Wait: add 1 bit to the header
  – Why is it enough?
Going faster: sliding window protocol

Still have the problem of keeping pipe full
- Generalize approach with > 1-bit counter
- Allow multiple outstanding (unACKed) frames
- Upper bound on unACKed frames, called window
How big should the window be?

How many bytes can we transmit in one RTT?
- $BW \text{ B/s} \times RTT \text{ s} \Rightarrow \text{"Bandwidth-Delay Product"}$
Maximizing Throughput

- Can view network as a pipe
  - For full utilization want bytes in flight $\geq$ bandwidth $\times$ delay
  - But don’t want to overload the network (future lectures)
Maximizing Throughput

- Can view network as a pipe
  - For full utilization want bytes in flight $\geq$ bandwidth $\times$ delay
  - But don’t want to overload the network (future lectures)
- What if protocol doesn’t involve bulk transfer?
  - Get throughput through concurrency – service multiple clients simultaneously
On reliable delivery
On reliable delivery

• Many link layer protocols don’t account for reliable delivery!
  – Eg. Wifi does, Ethernet does not

• Usually, reliable delivery guaranteed by other protocol layers if needed, such as TCP

• Why might we NOT want reliable delivery at the link layer?
Summary: Reliable delivery

• Want exactly once
  – At least once: acks + timeouts + retransmissions
  – At most once: sequence numbers

• Want efficiency
  – Sliding window
We did not cover these...
IP Checksum

- Fixed-length code
  - n-bit code should capture all but $2^{-n}$ fraction of errors
    - Why?
    - Trick is to make sure that includes all common errors
- IP Checksum is an example
  - 1’s complement of 1’s complement sum of every 2 bytes
    ```c
    uint16 cksum(uint16 *buf, int count) {
        uint32 sum = 0;
        while (count--)
            if ((sum += *buf) & 0xffff0000) // carry
                sum = (sum & 0xffff) + 1;
        return ~(sum & 0xffff);
    }
    ```
- Checking
  - Do the sum again, including the checksum. If correct, the sum should be all 1’s (This is super fast to check)
How good is it?

• 16 bits not very long: misses how many errors?
  – 1 in $2^{16}$, or 1 in 64K errors

• Checksum does catch all 1-bit errors

• But not all 2-bit errors
  – E.g., increment word ending in 0, decrement one ending in 1

• Checksum also optional in UDP
  – All 0s means no checksums calculated
  – If checksum word gets wiped to 0 as part of error, bad news
“This is a simple to compute checksum and experimental evidence indicates it is adequate, but it is provisional and may be replaced by a CRC procedure, depending on further experience.”
CRC – Error Detection with Polynomials

- Goal: maximize protection, minimize bits
- Consider message to be a polynomial in $\mathbb{Z}_2[x]$
  - Each bit is one coefficient
  - E.g., message 10101001 $\rightarrow m(x) = x^7 + x^5 + x^3 + 1$
- Can reduce one polynomial modulo another
  - Let $n(x) = m(x)x^3$. Let $C(x) = x^3 + x^2 + 1$.
  - $n(x) \equiv r(x) \mod C(x)$
  - Find $q(x)$ and $r(x)$ s.t. $n(x) = q(x)C(x) + r(x)$ and degree of $r(x) <$ degree of $C(x)$
  - Analogous to taking 11 $\mod 5 = 1$
Polynomial Division Example

- Just long division, but addition/subtraction is XOR
CRC

• Select a divisor polynomial \( C(x) \), degree \( k \)
  – \( C(x) \) should be irreducible – not expressible as a product of two lower-degree polynomials in \( \mathbb{Z}_2[x] \)

• Add \( k \) bits to message
  – Let \( n(x) = m(x)x^k \) (add \( k \) 0’s to \( m \))
  – Compute \( r(x) = n(x) \mod C(x) \)
  – Compute \( n'(x) = n(x) - r(x) \) (will be divisible by \( C(x) \)) (subtraction is XOR, just set \( k \) lowest bits to \( r(x) \))

• Checking CRC is easy
  – Reduce message by \( C(x) \), make sure remainder is 0
Why is this good?

• Suppose you send $m(x)$, recipient gets $m'(x)$
  – $E(x) = m'(x) - m(x)$ (all the incorrect bits)
  – If CRC passes, $C(x)$ divides $m'(x)$
  – Therefore, $C(x)$ must divide $E(x)$

• Choose $C(x)$ that doesn’t divide any common errors!
  – All single-bit errors caught if $x^k$, $x^0$ coefficients in $C(x)$ are 1
  – All 2-bit errors caught if at least 3 terms in $C(x)$
  – Any odd number of errors if last two terms ($x + 1$)
  – Any error burst less than length $k$ caught
Common CRC Polynomials

- Polynomials not trivial to find
  - Some studies used (almost) exhaustive search
- CRC-8: \( x^8 + x^2 + x^1 + 1 \)
- CRC-16: \( x^{16} + x^{15} + x^2 + 1 \)
- CRC-32: \( x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1 \)
- CRC easily computable in hardware
An alternative for reliability

• Erasure coding
  – Assume you can detect errors
  – Code is designed to tolerate entire missing frames
    • Collisions, noise, drops because of bit errors
  – Forward error correction
• Examples: Reed-Solomon codes, LT Codes, Raptor Codes
• Property:
  – From K source frames, produce B > K encoded frames
  – Receiver can reconstruct source with any K' frames, with K’ slightly larger than K
  – Some codes can make B as large as needed, on the fly
LT Codes

- Luby Transform Codes
  - Michael Luby, circa 1998

- Encoder: repeat B times
  1. Pick a degree d
  2. Randomly select d source blocks. Encoded block $t_n = \text{XOR or selected blocks}$
LT Decoder

- Find an encoded block $t_n$ with $d=1$
- Set $s_n = t_n$
- For all other blocks $t'_n$ that include $s_n$,
  set $t'_n = t'_n \oplus s_n$
- Delete $s_n$ from all encoding lists
- Finish if
  1. You decode all source blocks, or
  2. You run out of blocks of degree 1
Next class

• Link Layer II
  – Ethernet: dominant link layer technology
    • Framing, MAC, Addressing
  – Switching