Link Layer: Basic Techniques

Data Transmission

Link speed unit: bps

→ abstraction

→ ignore carrier frequency, coding etc.

Simplest case: point-to-point link

→ wired or wireless
Interested in *completion time*:

→ time elapsed between sending/receiving first bit
→ i.e., how long will it take?

• Single bit:
  → ≈ $L$/SOL (lower bound)
  → latency (or propagation delay)
  → optical fiber, wireless: exact

• Multiple, say $S$, bits:
  → ≈ $L$/SOL + $S/B$
  → latency + transmission time

Latency vs. transmission time: which dominates?

→ a lot to send, a little to send, ... 
→ satellite, Zigbee, WLAN, broadband WAN
Reliable Transmission

Main method: ARQ (Automatic Repeat reQuest)

→ use retransmission

→ used in both wired/wireless

• function duplication
  → link layer, transport layer, etc.

• alternative: FEC (forward error correction)
  → transmit redundant information

  → not assured

  → pros and cons?
ARQ: three components

- timer
- acknowledgment (ACK)
- retransmit
Special case: stop-and-wait

Handle one packet (i.e., frame) at a time.
Issue of RTT (Round-Trip Time) & timer management:

- what is proper value of timer?
  → RTT estimation

- easier for single link
  → RTT is more well-behaved

- more difficult for multi-hop path in internetwork
  → latency + queueing effect

A “good” thing about stop-and-wait:

→ simple throughput formula
Stop-and-wait throughput (bps):

- RTT
- frame size (bits)

\[ \text{throughput} = \frac{\text{frame size}}{\text{RTT}} \]

Another important problem: not keeping the pipe full.

\[ \text{delay-bandwidth product} \]
\[ \text{volume of data travelling on the link} \]

High throughput: want to keep the pipe full
Ex.: Link BW 1.5 Mbps, 45 msec RTT

- if frame size 1 kB, then throughput:
  \[ \Rightarrow 1024 \times \frac{8}{0.045} = 182 \text{ kbps} \]
  \[ \Rightarrow \text{utilization: only} \frac{182 \text{ kbps}}{1500 \text{ kbps}} = 0.121 \]

- note: delay-bandwidth product
  \[ \Rightarrow 1.5 \text{ Mbps} \times 45 \text{ msec} = 67.5 \text{ kb} \approx 8 \text{ kB} \]

What happens to utilization if RTT increases to 90 msec?

What happens if bandwidth increases to 3 Mbps?

\[ \Rightarrow \text{how to reduce bandwidth wastage?} \]
**Sliding Window Protocol**

→ send block (i.e., window) of data

**Issues:**

- Shield application process from reliability management chore
  → exported semantics: continuous data stream
  → simple app abstraction: e.g., `read` system call
- Both sender and receiver have limited buffer capacity
  → task: plug holes & flush buffer
Simple solution when receiver has infinite buffer capacity:

• sender keeps sending at maximum speed
• receiver informs sender of holes
  → “I’m missing this and that”
  → called negative ACK
• sender retransmits missing frames

Drawbacks?

What about positive ACK?

  → pros and cons
Sliding window operation with positive ACK:

- **SWS**: Sender Window Size (sender buffer size)
- **RWS**: Receiver Window Size (receiver buffer size)
- **LAR**: Last ACK Received
- **LFS**: Last Frame Sent
- **NFE**: Next Frame Expected
- **LFA**: Last Frame Acceptable
Assign sequence numbers to frames.

\[ \rightarrow \text{IDs} \]

Maintain invariants:

- \( \text{LFA} - \text{NFE} + 1 \leq \text{RWS} \)
- \( \text{LFS} - \text{LAR} + 1 \leq \text{SWS} \)

Sender:

- Receive ACK with sequence number \( X \)
- Forward LAR to \( X \)
- Flush buffer up to (but not including) LAR
- Send up to SWS \(- (LFS - LAR + 1)\) frames
- Update LFS
Receiver:

- Receive packet with sequence number $Y$
- Forward to (new) first hole & update NFE
  $\rightarrow$ NFE need not be $Y + 1$
- Send cumulative ACK (i.e., NFE)
- Flush buffer up to (but not including) NFE to application
- Update LFA $\leftarrow$ NFE + RWS − 1
Sequence number wrap-around problem:

\[ SWS < \frac{(\text{MaxSeqNum} + 1)}{2} \]

\[ \rightarrow \text{ why?} \]
\[ \rightarrow \text{ consider special case: stop-and-wait} \]
\[ \rightarrow \text{ is sequence number needed?} \]