**Direct Link Communication I: Basic Techniques**

**Data Transmission**

Link speed unit: bps

→ abstraction

→ ignore carrier frequency, coding etc.

Point-to-point link:

![Diagram of a point-to-point link](image)

→ wired or wireless

→ includes broadcast case
Interested in \textit{completion time}:

\rightarrow \text{ time elapsed between sending/receiving first bit}

- Single bit:
  \rightarrow \approx \frac{L}{SOL} \text{ (lower bound)}
  \rightarrow \text{ latency (or propagation delay)}
  \rightarrow \text{ optical fiber, wireless: exact}

- Multiple, say $S$, bits:
  \rightarrow \approx \frac{L}{SOL} + \frac{S}{B}
  \rightarrow \text{ latency + transmission time}

Latency vs. transmission time: which dominates?

\rightarrow \text{ a lot to send, a little to send, \ldots}

\rightarrow \text{ satellite, Zigbee, WLAN, broadband WAN}
Multi-hop link (generalize 2-hop case):

- Case 1: $B_1 = B_2$
  \[ \rightarrow 2 \left( \frac{L}{SOL} + \frac{S}{B} \right) + \varepsilon \]
  \[ \rightarrow \varepsilon: \text{processing overhead at intermediate node} \]
  \[ \rightarrow \text{minor detail: impact of packetization} \]

- Case 2: $B_1 < B_2$

- Case 3: $B_1 > B_2$
  \[ \rightarrow \text{without memory, i.e., buffer: information loss} \]
  \[ \rightarrow \text{loss rate} = 1 - \left( \frac{B_2}{B_1} \right) \text{ at full throttle} \]
  \[ \rightarrow \text{how much buffer space required for no loss?} \]
Reliable Transmission

Principal methodology: ARQ (Automatic Repeat reQuest)

→ use retransmission

→ used in both wired/wireless

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

• alternative: FEC
  → not assured
  → hybrid schemes
Three components:

- timer
- acknowledgment (ACK)
- retransmit
Stop-and-Wait

Assumption: Frame is “lost” due to corruption; discarded by NIC after error detection.
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
Another key problem: not keeping the pipe full.

→ delay-bandwidth product

→ volume of data travelling on the link

High throughput: want to keep the pipe full

Stop-and-wait throughput (bps):

• RTT

• frame size (bits)

→ throughput = frame size / RTT
Ex.: Link BW 1.5 Mbps, 45 ms RTT

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

- if frame size 1 kB, then throughput:
  \[ 1024 \times 8/0.045 = 182 \text{ kbps} \]

  \[ \rightarrow \text{utilization: only } 182 \text{ kbps}/1500 \text{ kbps} = 0.121 \]

Solution: increase frame size

- brute increase of frame size can be problematic
  \[ \rightarrow \text{bully problem} \]

  \[ \rightarrow \text{existing LAN frame standards (legacy compatible)} \]

- send blocks of data, i.e., sequence of frames
Sliding Window Protocol

⟶ send window/block of data

Issues:

• Shield application process from reliability management chore
  ⟶ exported semantics: continuous byte stream
  ⟶ simple app abstraction: e.g., read system call

• Both sender and receiver have limited buffer capacity
  ⟶ efficiency: space-bounded computation
  ⟶ task: “plug holes & flush”
Simple solution when receiver has infinite buffer capacity:

- sender keeps sending at maximum speed
- receiver informs sender of holes
  → i.e., negative ACK
- sender retransmits missing frames
  → sender’s buffer capacity?
  → need for positive ACK?

With finite buffer:

→ issue of bookkeeping

Flow control & congestion control:

→ sending too much is counterproductive
→ regulate sending rate
Set-up:

- **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 $Y$
- Forwind to (new) first hole & update NFE
  \[ \rightarrow \text{NFE need not be } Y + 1 \]
- Send cumulative ACK (i.e., NFE)
- Flush buffer up to (but not including) NFE to application
- Update LFA $\leftarrow NFE + RWS - 1$
ACK variants:

- piggyback
- negative ACKs
- selective ACKs

Sequence number wrap-around problem:

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

\[ \rightarrow \text{ note: stop-and-wait is special binary case} \]