Direct Link Communication: Technology and Access Control

Reliable Transmission

Principal methodology: ARQ (Automatic Repeat reQuest)

→ use retransmission

→ common to wired/wireless communication

• used at link layer and higher layers (e.g., TCP)

  → function duplication (needed?)

• alternative: FEC

  → not assured
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
  → hidden factor
- more difficult for multi-hop path in internetwork
  → propagation latency + queueing effect
Another key problem: not keeping the pipe full.

\[ \rightarrow \text{delay-bandwidth product} \]

\[ \rightarrow \text{volume of data travelling on the link} \]

High throughput: want to keep the pipe full

Stop-and-wait throughput:

- RTT
- frame size (bits)

\[ \rightarrow \text{throughput} = \text{frame size} / \text{RTT} \]

\[ \rightarrow \text{exact?} \]
**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 is problematic
  
  \[ \rightarrow \text{why?} \]

- send blocks of data, i.e., sequence of frames
  
  \[ \rightarrow \text{creates management problem} \]
Sliding window protocol

Issues:

- Shield application process from reliability management chore
  → exported semantics: continuous byte stream
  → application view (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
- sender retransmits missing frames

→ sender does not need to wait

→ what about sender’s buffer capacity?
More complex solution due to bounded space:

- Sliding window technique
  → local repair with positive/negative ACK
- Flow control & congestion control
  → sending too much is counterproductive
  → regulate sending rate
Set-up:

- **SWS**: Send Window Size
- **RWS**: Receiver Window Size
- **LAR**: Last ACK Received
- **LFS**: Last Frame Sent
- **NFE**: Next Frame Expected
- **LFA**: Last Frame Acceptable
Assign sequence number to individual frames.

Maintain invariants:

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

\[ \rightarrow \text{why less-than-or-equal?} \]

Sender: update LAR; flush buffer up to LAR; send more frames; update LFS

Receiver: send cumulative ACK; flush buffer up to ACK to application;

- \( \text{NFE} \leftarrow \text{SeqNumToAck} + 1 \)
- \( \text{LFA} \leftarrow \text{SeqNumToAck} + \text{RWS} \)

SeqNumToAck denotes the largest sequence number not yet acknowledged ("first hole")
ACK variants:

- piggyback
- negative ACKs
- selective ACKs

Sequence number wrap-around problem:

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

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