Software Protocols

Protocol - A methodology or discipline for endpoints to use in communicating across a channel.

A channel could be a physical layer link or it could be a virtual entity with explicity defined properities.

Endpoints are usually processes executing on a system.

Protocol Properties

- Connectionality - how the endpoints maintain the relationship.
- Reliability - how the protocol manages errors.
  - Data errors
  - Lost packets
  - Duplicate packets
  - Out-of-order packets
- Flow Control - how are data rate differences managed.
- Directionality - simplex or duplex.
- Addressing - how do protocols identify endpoints.

Simplex Unreliable Protocol

Simplex Unreliable Protocol

- No error checking nor response.

An SUP uses 10 bytes of header and 90 bytes of data in each message. Propogation delay and processing time is 0.1 s and the data rate is 1 Mbps,

$$U = \frac{90/10^6}{100/10^6+0.1} = 0.9$$

Stop and Wait Protocol

Stop and Wait Protocol

- Still simplex in that data only goes one direction although in precise terms, the communication system is duplex.
- Will detect data errors and possibly lost messages.
- Negative Acknowledgements - NAK, to indicate a received message had an error.
- Algorithm
  1. Wait for a message to send.
  2. Send message
  3. Wait for the ACK, ...

Performance

$$U = \frac{DataTime}{TotalTime}$$

- $Message\ time = data\ time + header\ time$
- $Total\ time = Message\ Time + propogation\ time + processing\ time$

Typically, the propogation time and processing time are lumped together so that C is the total time to send a message and get a response.

$$U = \frac{M/R}{L/R + C}$$

However, times are usually not dependent so much on data rate as on overall network and local system performance. The critical issue for software protocols is high utilization in the face of unpredictable underlying performance and meeting reliability constraints.

Automatic Request Protocols

- We need a timer to allow the sending side to escape from the wait.
- Algorithm
  1. Wait for a message to send.
  2. Send message
  3. Set timer
  4. Wait for the ACK or a timer expiration
  5. If the timer expires, go to 2.
  6. If the ACK arrives, stop the timer and go to 1.

Performance

$$U = \frac{DataTime}{TotalTime}$$

- $Message\ time = data\ time + header\ time$
- $Total\ time = Message\ Time + Network\ latency + Resend\ Costs$

The resend cost is the time it takes for the timer to expire plus the time to send the message. The network latency only gets counted once. Let T be the timeout value and A the probability of a successful send. Then let E be the expected number of failures, where $E = (1/A - 1)$,

$$U = \frac{M/R}{L/R + C + E(T + L/R}$$

Handling Message Order Errors

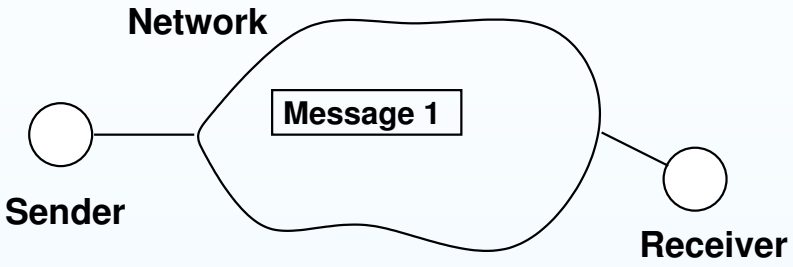
Sender:

1. $SEQ = 0$.
2. Wait for a message to send.
3. $SEQ = (SEQ + 1) \, mod \, 2$,
4. Send packet with SEQ, and set the ACK timer.
5. If the timer expires, go to 4.
6. If the $ACK = SEQ1$ arrives, stop the timer and go to 1.

Receiver:

1. $NFE = 0$.
2. Wait for a message to arrive.
3. If no errors and $NFE = SEQ$, send ACK with NFE
4. $NFE = (NFE + 1) \, mod \, 2$, go to 2.

The Delayed Packet Problem



- Message is delayed in the network.
- The timer expires and message 1 is sent again.
- Message 1 is acknowledged.
- Message 0 is sent and acknowledged.
- A new Message 1 is sent and is lost.
- The old Message 1 is arrives.

The Delayed Packet Problem

How many sequence numbers are needed to avoid this problem?
If a sender can send N messages per second and the maximum delay is K seconds, we need $N \times K$ sequence numbers.