OSI Data Link Layer

Part of Network Access Layer of TCP/IP

Details Computer-to-Network Issues

Data Link Issues

- Currency of Data-Link is frames or packets.
- Issues to be addressed here are…
  - Framing
  - Error Control
  - Flow Control
- We have seen how frame and error detection are done, now we have to *do* something about lost or damaged frames.

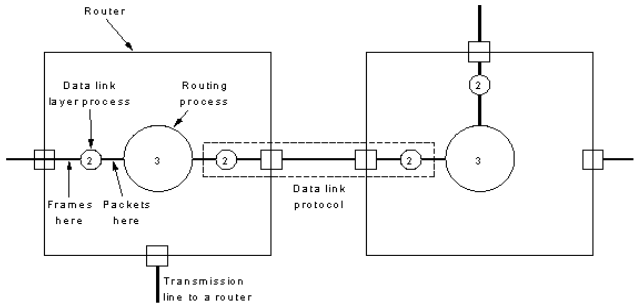
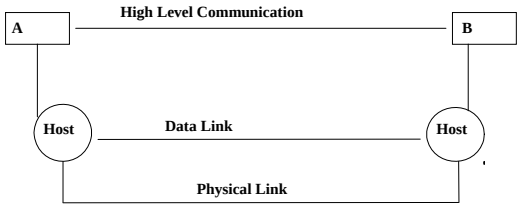


Fig. 3-2. Placement of the data link protocol.

The Need For Protocols

- Protocols are algorithms which will implement framing, flow control and error detection and correction.
- Makes error prone physical layer appear as error free to higher layers.
- Does so as efficiently as possible using valuable bandwidth.

Two Hosts Communicate



Idea for a Protocol

- To begin designing a protocol let us make the following *unrealistic assumptions* :
  - a) Simplex data transmission.
  - b) Transmitting and receiving hosts are always ready to transmit and receive data.
  - c) Processing time is negligible and infinite buffer space is available
  - d) Perfect error-free link

Utopia Protocol

- Make all assumptions, a, b, c, d.
- For this protocol the control header and checksum are unnecessary. The transmitting host simply takes packets from host *A* (which always has one ready) and pumps them as fast as it can onto the physical link.
- The receiver accepts the frames and passes them straight to host *B*.

Stop and Wait

- Drop assumption c); processing time $\neq 0$; buffer $\neq \infty$; not simplex
- In practice the receiving host needs time to process incoming frames, has only a finite amount of buffer space to queue processed frames i.e. the receiver needs to be able to prevent the sender from flooding it with data faster than it can handle it. Some form of handshaking is required.

Stop and Wait (cont.)

- In a simple stop-and-wait protocol, the receiver sends an acknowledgement frame back to the sender after delivering the packet to the host. Only after receiving the *control frame* will the sender fetch and transmit the next packet.

Positive Acknowledgement with Retransmission PAR

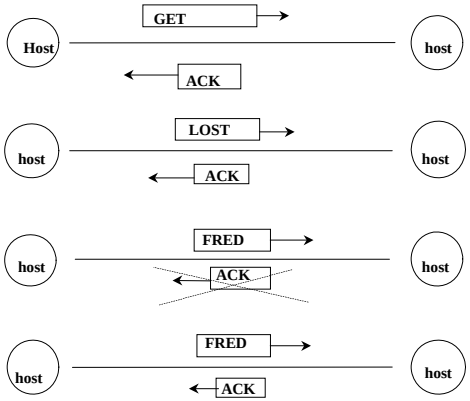
- Noisy channel simplex protocol: drop assumptions c) and d)
- With error prone physical link, frames may be either damaged or lost completely.
- However,
  - Damaged frames can be detected by the checksum.
  - Lost frames will not be acknowledged. Eventually the sender will tire of waiting for an acknowledgement, timeout, and retransmit the frame.

Bright Idea for a Protocol

- Same as stop-and-wait, except that damaged frames are not acknowledged, causing a timeout and subsequent retransmission.

Lets Examine a Protocol

- Suppose the message “Get Lost Fred” is being sent from *A* to *B*, one word per packet



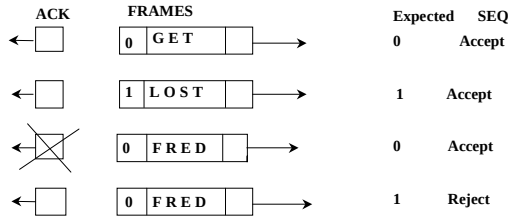
Get Lost Fred

- *B* receives “Get Lost Fred Fred” and the protocol has failed !
- Basically what has happened is that the receiver has accepted a duplicate frame.
- The solution is to use a *SEQ* sequence number in the control header to differentiate between frames and allow duplicates to be discarded.

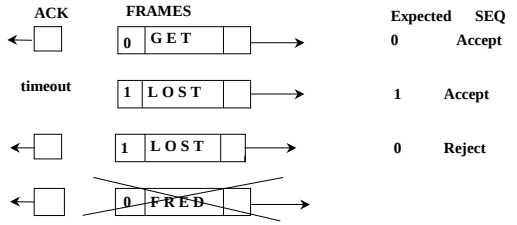
Get Lost Fred

- As a frame must be acknowledged before the next one is sent, a one-bit sequence number [0,1] is sufficient. The receiver will expect alternatively numbered frames (0 1 0 1 0 1 ... etc.). Any frame with the wrong sequence number is rejected as a duplicate (but still acknowledged).

Get Lost Fred



Premature timeouts

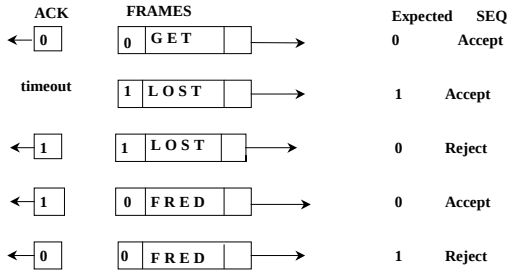


- The message “Get Lost” is received and the protocol has failed

Solution

- Include in the ACK field of the acknowledgement control frame the SEQ number of the last frame *received without error*. Then if this number (0 or 1) differs from the transmitted frame, the sender transmits it again.
- The resulting PAR simplex protocol will now work in the face of any combination of garbled frames, lost frames and premature timeouts.

Solution



Summary

- After transmitting a frame and starting the timer, Host *A* waits for a response. There are three possibilities:
  - an acknowledgement frame arrives undamaged
  - a damaged acknowledgement arrives, or
  - the timer goes off.

Summary (cont.)

- If a valid ACK comes in, *A* fetches the next packet and puts it in the buffer, overwriting the previous packet, and advancing the sequence number. If a damaged frame arrives or no frame at all arrives, neither a buffer nor the sequence number are changed, so that a duplicate can be sent.

Bi-directional PAR

- The control header of all frames contains the three fields *KIND*, *SEQ* and *ACK*.
- Sending data packets/acknowledgements in both directions is no problem -- by looking at the *KIND* bit in the header, the receiver knows which it is dealing with.

- However, this would be inefficient.
- Consider an Host *B* which is about to acknowledge a data frame received from Host *A*, and also about to send off a data frame to *A*.
- Instead of sending two frames, the acknowledgement can hitch a lift on the data frame, using the ACK field in the header.
- This is called *piggybacking*.

- As we are still making assumption b), all acknowledgements can be piggybacked. Thus, data packets are bounced back and forth between *A* and *B*.
- Notes:
  - For the protocols considered so far, only one frame is *in the wire* at any one time.
  - The sending IMP needs to keep a copy of each frame in a buffer for possible retransmission until the frame has been successfully acknowledged.

- Assumption b) is easily dealt with. If there is no outgoing data frame, the host will wait a short while to see if one comes along to provide a piggyback. If not, a separate acknowledgement frame will be sent.
- It must not wait too long to avoid unnecessary duplicates being sent due to the sender timing-out.

- Up until now, lost and damaged frames have been dealt with in the same way. No *ACK* is sent, leading to timeout and retransmission. The timeout period is usually set quite long in order to avoid complications caused by premature timeout. This is inefficient, as while the timeout is expiring, the link is not being used.
- A partial solution is *NAK*, a negative acknowledgement. This is sent immediately a damaged frame is received and illicits immediate retransmission.

- The *NAK* may also be piggybacked.
- If the *NAK* is damaged, no harm is done as the sender will eventually timeout and retransmit as before.
- A damaged frame is *Nak*'d only once.

Pipelining

- When propagation delay is not negligible, these previous methods are wasteful of bandwidth.
- The solution is to '*fill up the pipe*'. However, doing this entails sending off frames before ACKs for previous frames have arrived.

Sliding Window Protocol

- Each outbound frame is given a sequence number in the range of 2^n-1 using an n-bit field, e.g. if n=1, then range is 0.....1 as in ABP or PAR protocols.
- Both sender and receiver keep *windows* informing them of which frames can be validly sent and which validly received.

Rules

- **Sender** :- The upper edge of sender is advanced when a frame is sent (up to max. window size). The lower edge advanced when ACK received for lowest numbered frame in the window.
- **Receiver** :- Both edges are advanced when the lowest numbered frame in window is correctly received and ACK sent.

Notes

- Buffering requirements at both sender and receiver depend on the size of the sending and receiving windows respectively.
- Each transmitted frame has its own separate timeout clock.

Notes (cont.)

- In these protocols, an acknowledgement for frame *N* is accepted as acknowledging all transmitted frames numbered up to *N* (counting circularly).
- Thus, if ACK(0) and ACK(1) were both destroyed, but ACK(2) now arrives, it implicitly acknowledges 0 and 1 also.

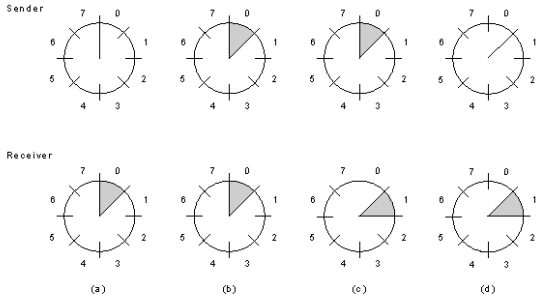
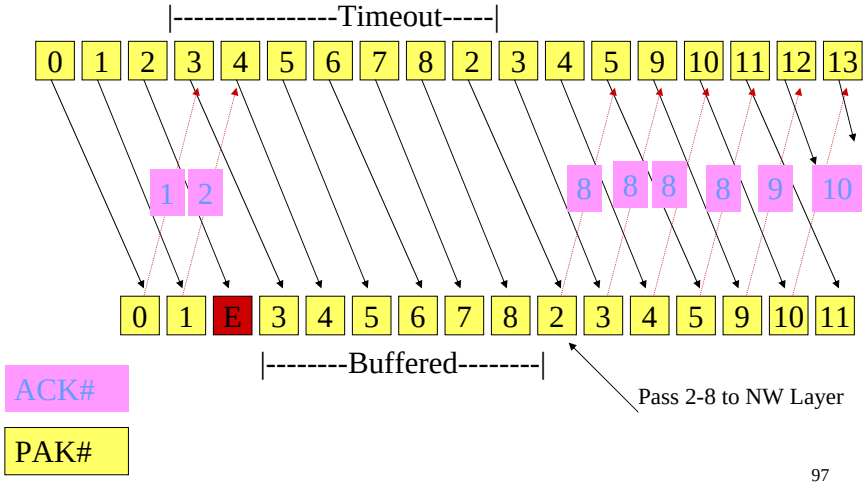


Fig. 3-12. A sliding window of size 1, with a 3-bit sequence number. (a) Initially. (b) After the first frame has been sent. (c) After the first frame has been received. (d) After the first acknowledgement has been received.

Example Session with Recovery



The Stopping Problem

- A Data-Link cannot be stopped.
- Consider a session termination.
- Neither terminal knows that other has sent *last* packet, the *last* packet must be ACK'd.
- In practice the data-link is dropped after the link is sensed as being dead for a prolonged period.

Protocol Verification

- To prove that a protocol works under any conditions is difficult.
- One approach is to use computer simulation and Formal Description Techniques (based on Finite State Machines).
- Computer software simulates possible paths through the FSM for the protocol.
- FDTs include Estelle, SDL and LOTOS.