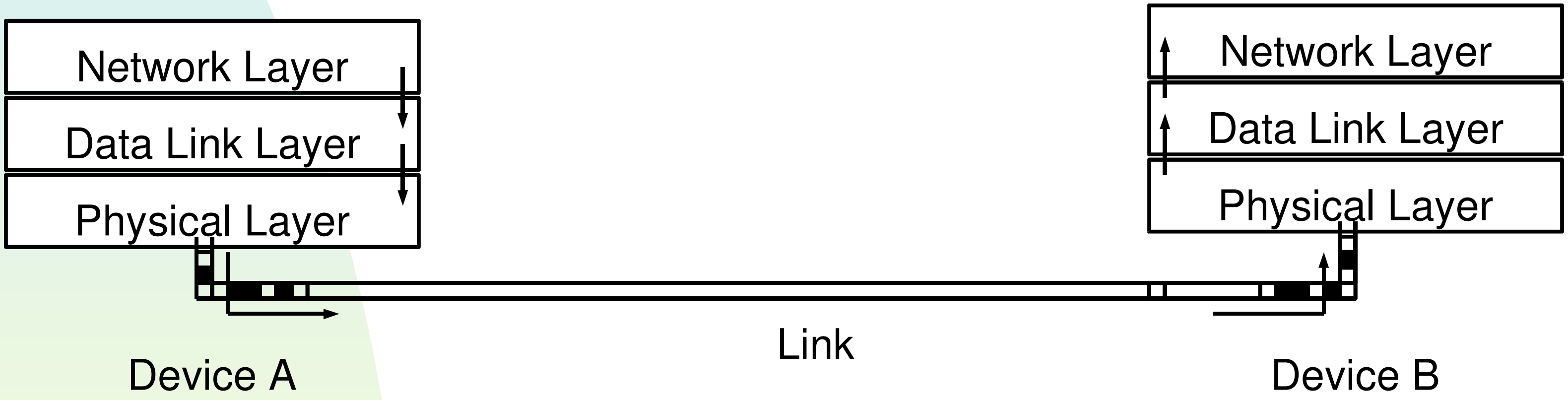
Data Link Protocols

- A data link protocol is the set of rules used to send data over an individual link from one host to the next.
  - Data link protocols vary from simple asynchronous transmission to reliable sophisticated frame based communications.



- Typically the Network Layer passes packets to the Data Link Layer. The Data Link Layer encapsulates the packets in frames and gives the frames to the Physical Layer for transmission.

Data Link Protocols

- The algorithm that controls the data link protocol resides in the Data Link Layer.

- It makes use of various function calls in order to interact with the layers above and below it.

wait_for_event(event_type *event)	Halt process until event occurs (event code passed back).
from_physical_layer(frame *f)	Pass up received frame.
to_physical_layer(frame *f)	Send frame.
from_network_layer(packet *p)	Pass down data packet.
to_network_layer(packet *p)	Send packet to Network Layer.

General Functions

- There are a number of other useful functions:

start_timer(seq_num k)

- Start countdown to *timeout event k* (timeout period is a system parameter).

stop_timer(seq_num k)

- Stop timer *k*.

start_ack_timer(void)

- Start countdown to *ack_timeout* event.

stop_ack_timer(void)

- Stop countdown to *ack_timeout* event.

disable_network_layer(void)

- Forbids *network_layer_ready* events.

enable_network_layer(void)

- Allow *network_layer_ready* events.

inc(k)

- Add 1 to *k* unless *MAX_SEQ,* in which case reset *k* to 0

The Frame Structure

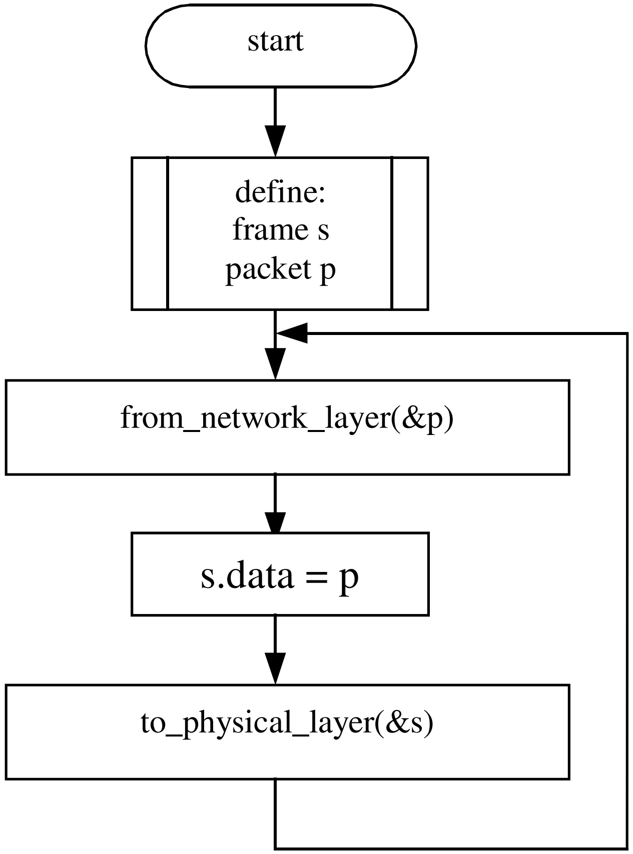
- The format of the frame structure typically looks something like this:

frame_kind kind	**seq_num seq**	**seq_num ack**	**packet data**

  - There can be three *kind*s of frame: *data_frame*, *ack_frame* and *nak_frame*. The last two types do not carry data.
  - For data frames, the packet of data passed down from the Network Layer is put in the *data* field of the frame structure.
  - In order to make sure that the packets are reassembled in the right order, a sequence number is placed in the *seq* field of each frame.
  - For reliable communications, it is important that correctly received frames are acknowledged. The sequence numbers of correctly received frames go into the *ack* field of the frame structure.

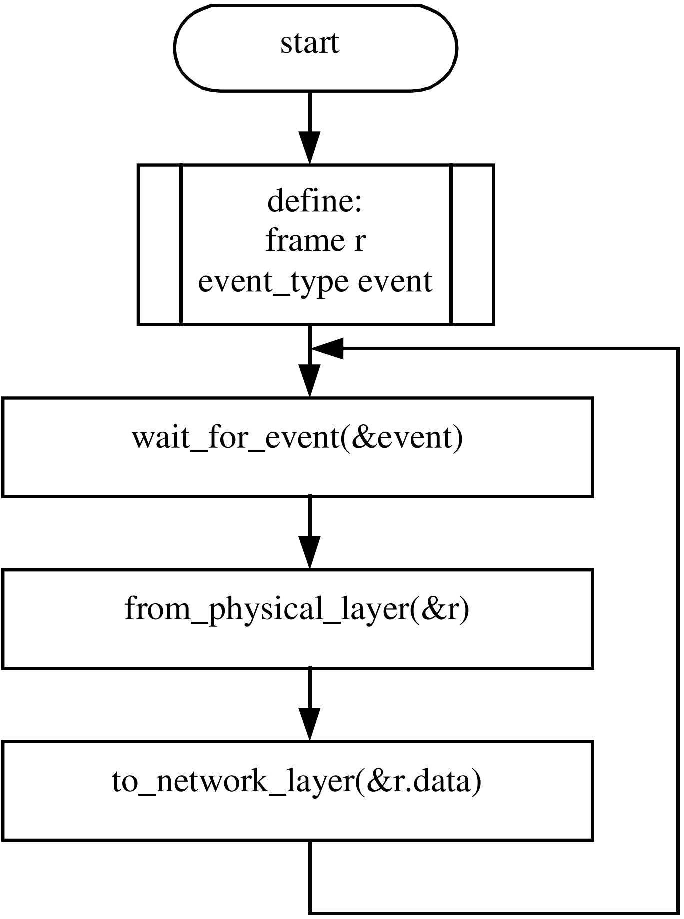
An Unrestricted Simplex Protocol

- Here is a simple algorithm for sending data over a noise free link. We are assuming that there is no transit delay and that the destination host has infinite buffer space.
  - We start of by defining the data structures we need to store the packet and frame data.
  - We get a packet from the network layer and place it in the data field of the frame structure.
  - Lastly, we pass the frame to the physical layer for transmission.
  - Then we do it all over again.



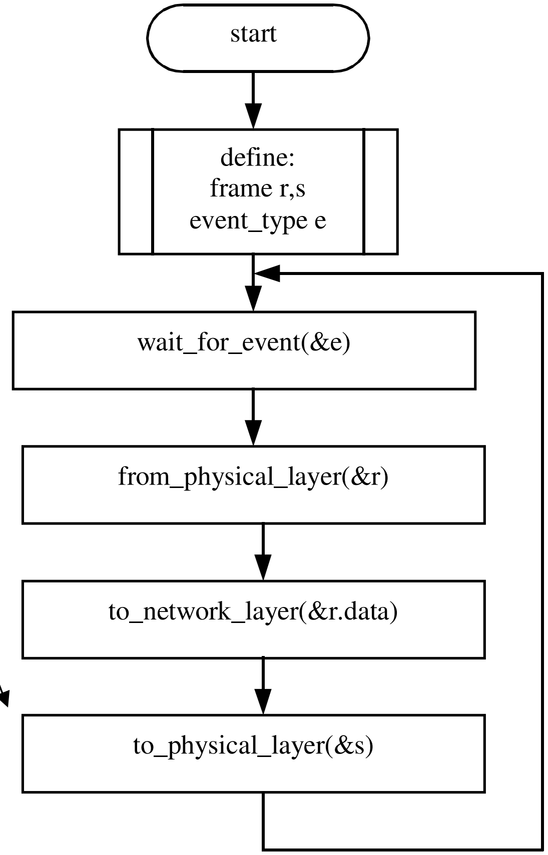
An Unrestricted Simplex Protocol

- Data is only sent in one direction using this protocol. The data is received by the destination host which uses the following algorithm:
  - Once again, we start by defining the various data structures we need to hold the data.
  - Then we go into a loop in which we wait for a frame arrival event.
  - When it happens, we get the frame from the physical layer.
  - We then pass it up to the network layer.
  - Then we wait for the next frame.



A Simple Stop-and-Wait Protocol

- The network layer in the receiving host may not always have enough buffer space to cope with all the frames .
  - We need to be able to tell the sending host to pause until the receiving host is ready. This is called *flow control*.
  - The destination host still works the same way except it now sends an empty frame back to the sending host.
  - This empty frame is used to tell the sending host that the receiving host is ready to accept another frame.



start

define:
frame r,s
event_type e

wait_for_event(&e)

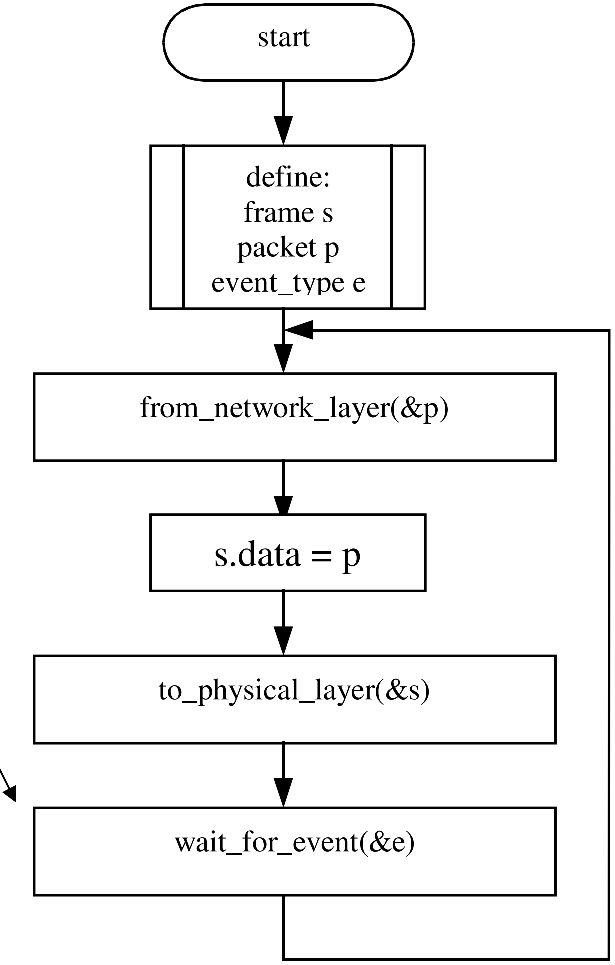
from_physical_layer(&r)

to_network_layer(&r.data)

to_physical_layer(&s)

A Simple Stop-and-Wait Protocol

- The sending host now only sends one frame at a time. It will not send another frame until it gets an empty frame back from the receiving host.
  - The algorithm works in exactly the same way as the first simplex algorithm except it now waits for an empty frame sent back from the receiving host.
  - Only when the empty frame arrives will the sending host send another frame.



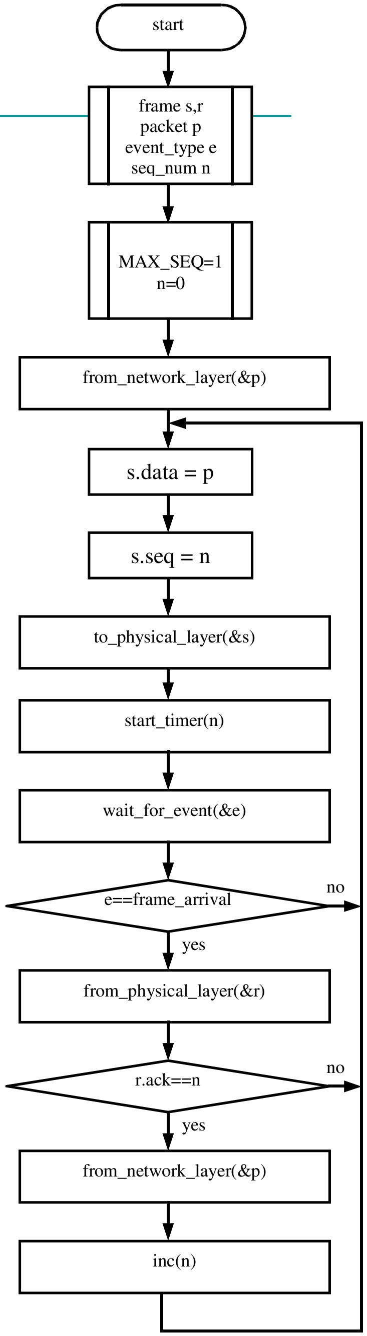
A Simple Protocol for a Noisy Channel

- Things become tricky when we take noise into account. Noise can damage the data in frames. This is usually discovered when the frame checksum (FCS) is tested.
  - For simplicity, we will assume that the frame checksums will automatically be checked on arrival and if an error is discovered then it will generate a *chksum_err* event.

- Each frame is given a sequence number (either 0 or 1). The sequence number alternatives for successive frames.
  - When a frame is correctly received, its sequence number is acknowledged by sending it back in the *ack* field of an empty frame.
  - When a frame is incorrectly received, no acknowledgement is sent back and the sender *times out*.

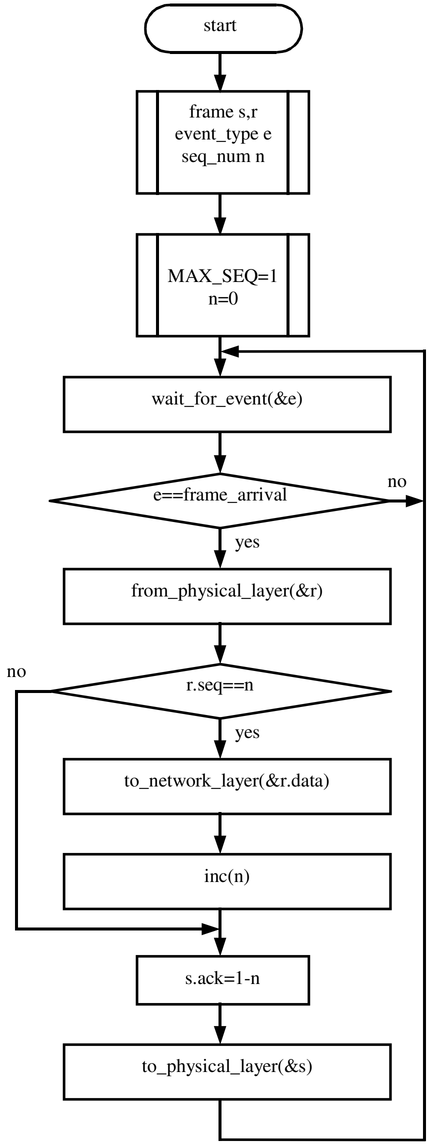
A Simple Protocol for a Noisy Channel

- The sequence number of the first frame is set to 0.
- It is transmitted and then a timer is started.
- Sometimes a frame may be completely lost so there may be nothing to acknowledge.
- If no acknowledgement is received by the time a timeout event occurs, the sender assumes that its frame was lost completely and resends it.
- If an acknowledge is received but has the wrong sequence number, the sender assumes that its frame was not received and resends it.
- If an acknowledge is received and the sequence number is correct, the sender sends the next frame (with a new sequence number).



A Simple Protocol for a Noisy Channel

- The algorithm for the receiving host looks like this:
  - The receiving host waits for a frame arrival.
  - If the frame was damaged, it waits for the sending host to timeout and send the frame again.
  - If the frame is alright then it gets the frame and checks its sequence number.
  - If the sequence number is alright then the packet is passed to the network layer and an acknowledgement is sent back to the sender.
  - If the sequence number is wrong, the wrong sequence number is sent back.

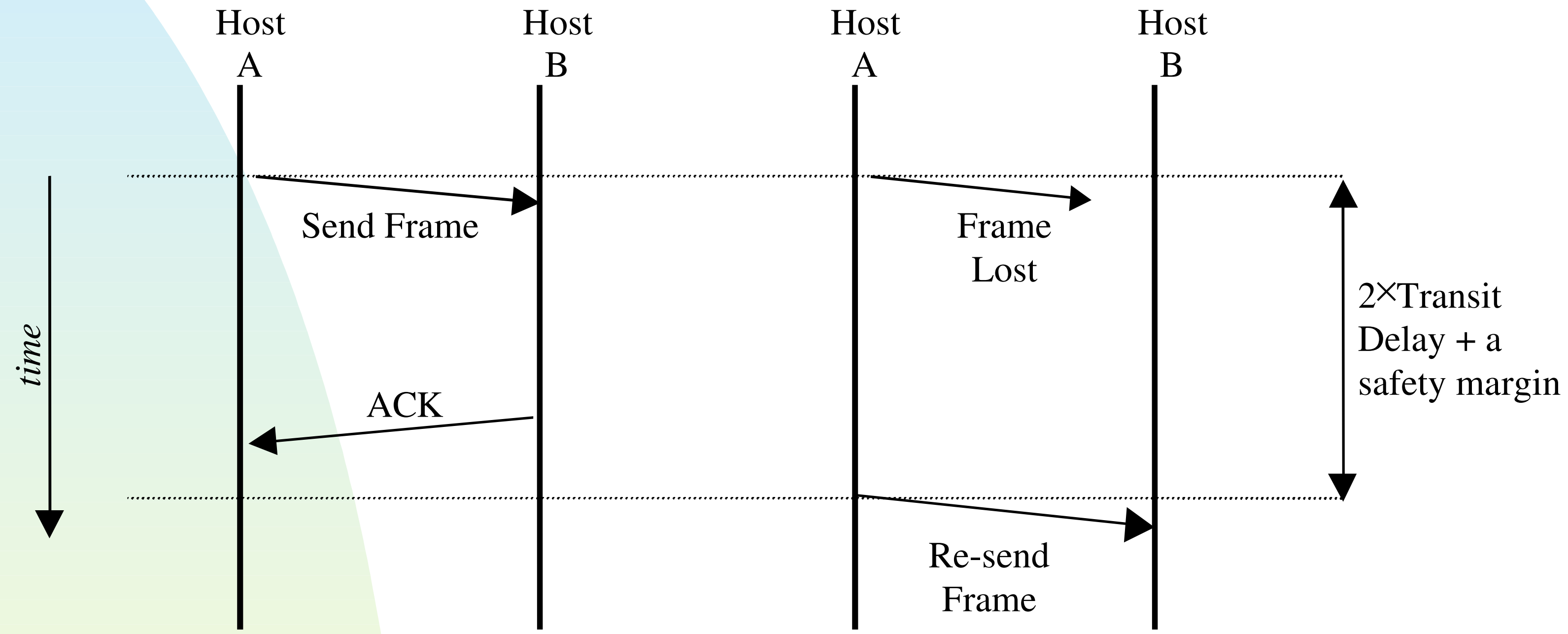


Timeout

- Sometimes frames can be damaged by noise so much that they are lost completely.
  - We cannot assume that any frame (that includes acknowledgement frames) will arrive.
  - If an acknowledgement for a particular frame does not arrive back in a reasonable time, the sender should assume the worst case scenario - that it’s data never made it to the receiving host.
  - By a *reasonable time*, we usually mean the time it would normally take to send a frame and receive an acknowledgement (twice the transit delay plus a small margin of safety to allow for the time it takes for a whole frame to be received and processing time).
  - After this time, the sender should retransmit the frame.

Timeout

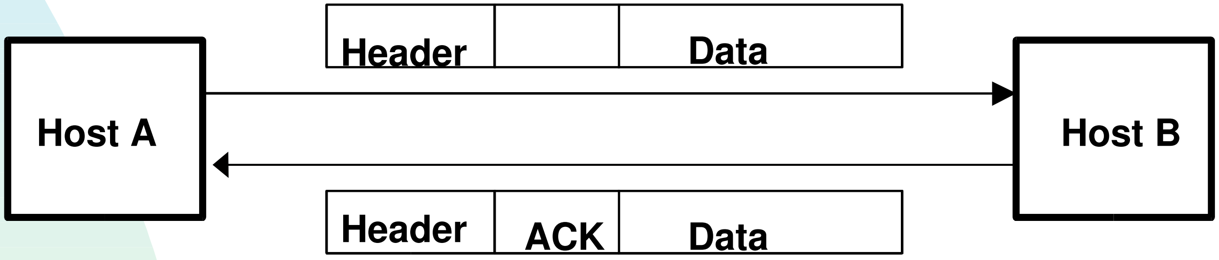
- The timeout mechanism can be shown using a *time sequence* diagram:



  - Left: sender successfully sends frame and receives an acknowledgement.
  - Right: frame is lost in transit. No acknowledgement arrives back in a reasonable time so frame is retransmitted.

Duplex Communication

- The protocols we have seen so far just send data in one direction. Of course, data will usually flow in both directions.



  - If frames are travelling in both directions anyway, it makes sense to *piggyback* acknowledgements on these frames.
  - This saves the overhead involved in sending a separate frame just to carry an acknowledgement.
  - Of course, if no frames are due to be sent back, we will have no choice but to send back a separate acknowledgement frame.

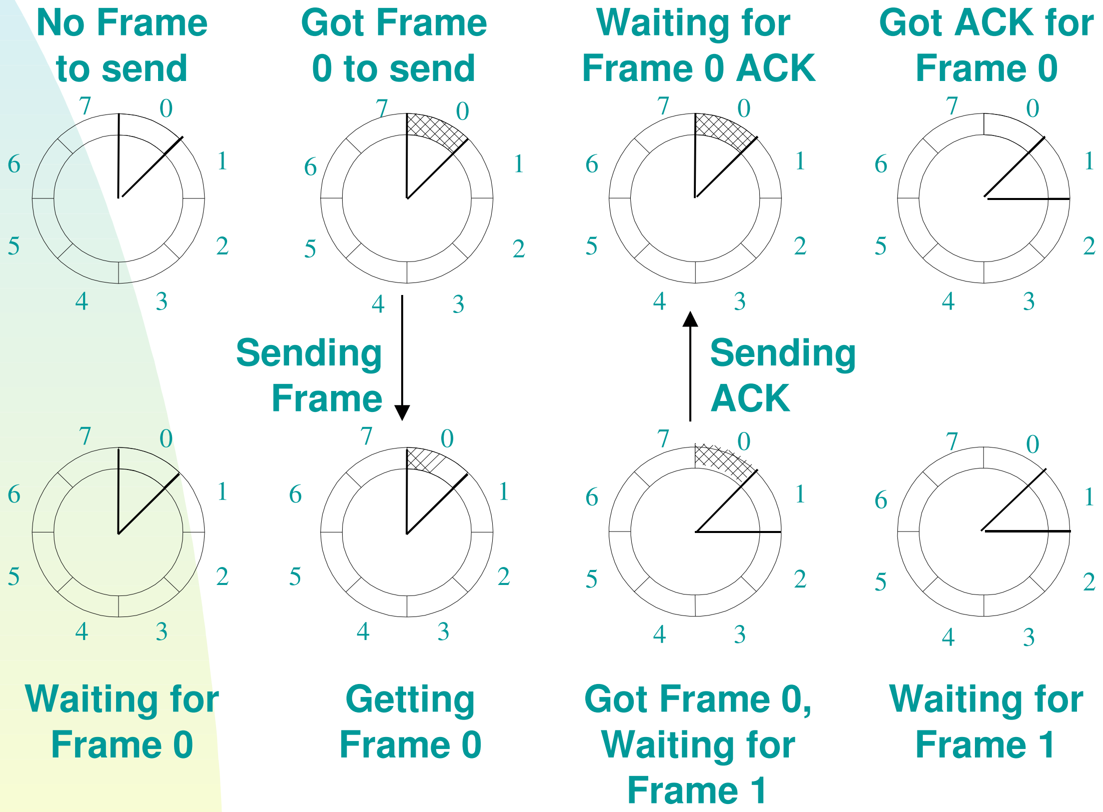
Sliding Windows Protocol

- To keep track of the frames we use a technique called *sliding windows*.
    - Both sending and receiving hosts maintain a range of sequence numbers called a *window*.
    - The sender sends only the frames with sequence numbers in its window. The window can only move on when the first sequence number in the window has been acknowledged.
    - The receiver only receives frames with sequence numbers in its window (any other frames are ignored). Its window can only move on when it receives the frame with the first sequence number in its window.

- The sliding window technique restricts the number of frames that can be outstanding, which is good for flow control.

Sliding Windows Protocol

- Here is what happens if we have a 3-bit sequence number and a window size of 1 on both sender and receiver:



No Frame to send

Got Frame 0 to send

Waiting for Frame 0 ACK

Got ACK for Frame 0

Sending Frame

Sending ACK

Waiting for Frame 0

Getting Frame 0

Got Frame 0, Waiting for Frame 1

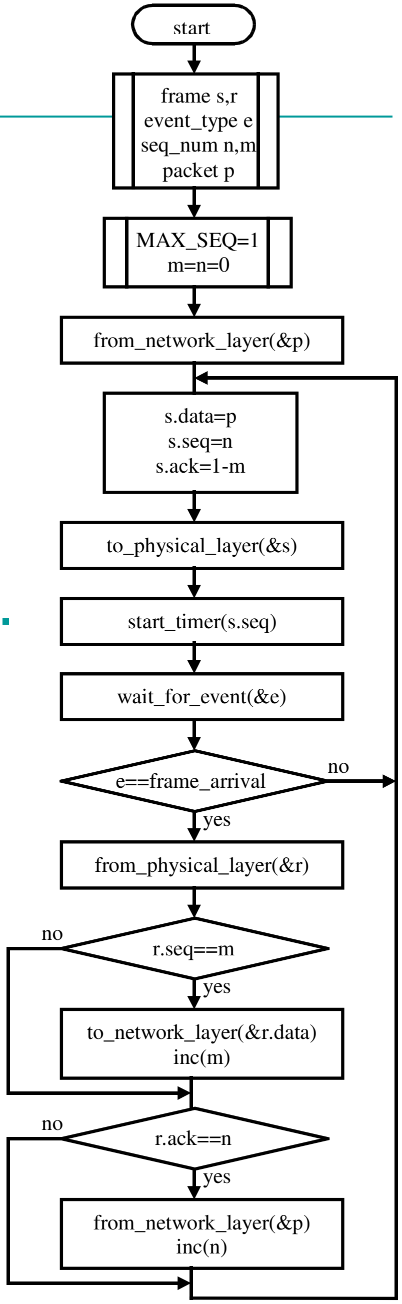
Waiting for Frame 1

Sequence Numbers

- Sequence numbers usually occupy a field with a limited number of bits in the frame format.
  - We do not want to have sequence numbers that are too large because that would waste valuable bandwidth.
  - The number of sequence numbers needs to be at least one greater than the size of the largest window intended to be used (a sliding window protocol can only work reliably if there is at least one sequence number outside the window!).
  - Because sequence numbers occupy a whole number of bits, the range of sequence numbers is usually 0 to 2^m-1 where *m* is the number of bits used.
  - For example, we could have a 1-bit sequence number. This will be enough for a sliding window protocol with window size 1.

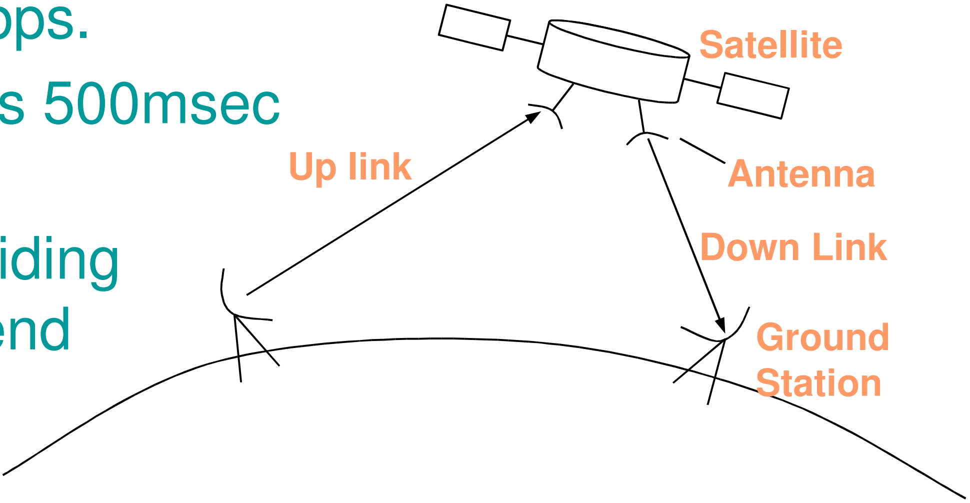
One-bit Sliding Window Protocol

- This algorithm works as both sender and receiver.
  - Sequence numbers are either 0 or 1.
  - Packets are put in the frame structure along with its sequence number and the sequence number of the last frame to be acknowledged.
  - The frame is sent and the timer started.
  - If a new frame arrives, check to see if its sequence number is in the window.
  - If not, then ignore its contents.
  - Check the acknowledgement field and if it acknowledges the last frame we sent then advance our window.
  - If any problems, re-send our last frame otherwise send the next frame.



Long Transit Delays

- The one-bit sliding window protocol will work fine over short half-duplex links.
- It is not so good over links with long transit delays. Imagine a link that uses a geo-stationary satellite:
  - The data rate is 50-kbps.
  - The round-trip delay is 500msec (that’s half a second).
  - If we used the 1-bit sliding window protocol to send 1000-bit frames, the receiver will receive the <u>whole</u> frame 270msec later.



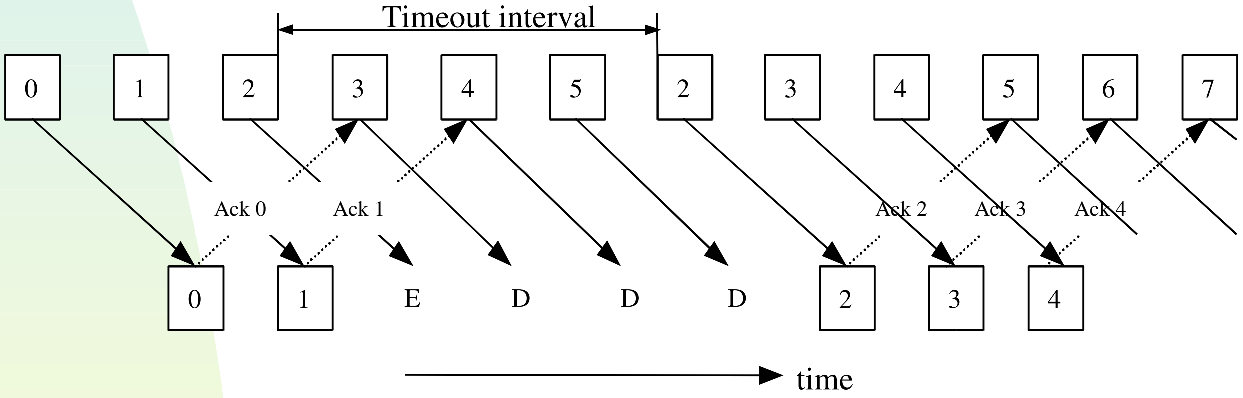
  - The acknowledgement will take a further 250msec to get back. Out of 520msec, data is only being sent for 20msec.
  - Only 20/520 = 4% of the link’s capacity is being utilised.

Long Transit Delays

- Rather than wasting so much channel capacity, a protocol can be designed to send enough frames to keep the link working at full capacity.
  - Rather than just allowing one outstanding frame (like the one-bit sliding window protocol does) we can allow up to 26 outstanding frames (enough to fill the 520msec round-trip delay period).
  - The sliding window of the sender need only be enlarged to 26. To accommodate this, a 5-bit (0-31) sequence number would be most suitable.
  - With a sliding window size of 26, up to 26 frames and acknowledgements can be in transit at any time.
  - The real question is: “What happens when a frame is damaged during transit?”

Go Back n

- One approach is to discard the damage frame and all the frames that follow it.
  - Eventually the sender will realise it has not received an acknowledgement for the damage frame within the timeout period. The sender will then retransmit the damage frame and all the frames that follow it.



  - The above time sequence diagram shows what happens when an erroneous frame (E) is received. All following frames are discarded (D) until the new copy of the erroneous frame is received.

Go Back n

- The main advantage of *go back n* is that the receiver only needs to maintain enough buffer space for one frame at a time.
    - Packets must be passed up to the Network Layer in the order in which they were sent by the Network Layer in the source host. Only one packet needs to be stored at a time since the frames will always arrive in the correct order.

- To implement the *go back n*, the sliding window of the receiver is set to 1 (i.e. different from the sliding window of the sender).
    - With a sliding window size of 1, the receiver will only accept the frame with the next expected frame sequence number. When a frame is damaged, the next expected frame sequence number remains the same.

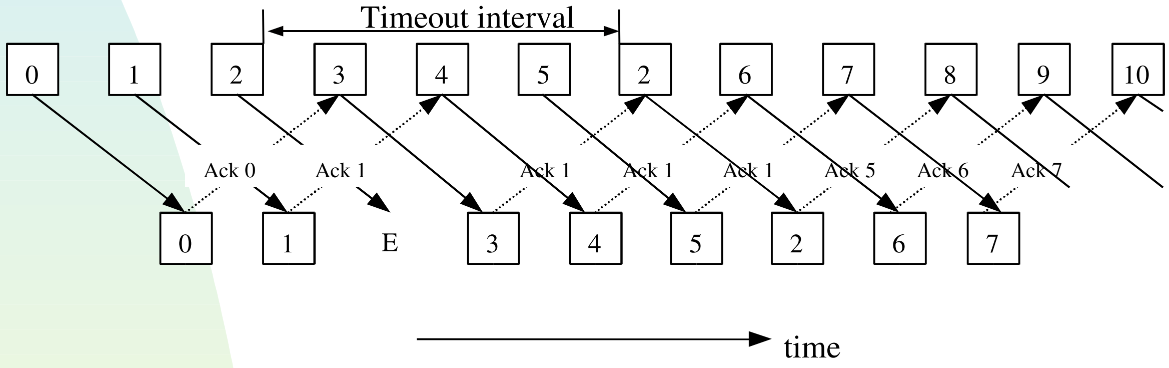
Selective Repeat

- *Go back n* can waste a lot of link capacity by re-sending frames that arrived perfectly the first time.
  - If a link is particularly noisy, this waste can significantly reduce the link's throughput.

- With *selective repeat*, only those frames that are damaged are re-sent. Undamaged frames are stored until they can be passed (in the correct order) to the Network Layer.
  - The receiver must have enough buffer space available to potentially store all the outstanding frames from the receiver.
  - Much less of the link's capacity is wasted since only the damaged frames are retransmitted and the rest of the time the link can carry useful data.

Selective Repeat

- Frames may arrive at the destination out of order. This means that the Data Link Layer is responsible for ordering them before sending the packets they contain to the Network Layer.



- The *selective repeat* protocol can be implemented by setting the receiver sliding window size to more than 1 (usually it is set to the same size as the sender's sliding window).