3.3. Elementary Data Link Protocols:

When a frame arrives at the receiver, the hardware computes the checksum. If the checksum is incorrect , the data link layer is so informed (event = cksum_err).

- If the inbound frame arrived undamaged, the data link layer is also informed (event = frame_arrival) so that it can acquire the frame for inspection using from_physical_layer.

- As soon as the receiving data link layer has acquired an undamaged frame, it checks the control information in the header, and if everything is all right, passes the packet portion to the network layer. Under no circumstances is a frame header ever given to a network layer.

- There is a good reason why the network layer must never be given any part of the frame header: to keep the network and data link protocols completely separate.

- Following slide shows some declaration common to many data link protocols:

- Five data structures are defined there:

1. **boolean,**
2. **seq_nr,**
3. **packet,**
4. **frame_kind,**
5. **frame**.

- A **boolean** is an enumerated type and can take on the values true and false.
- A **seq_nr** is a small integer used to number the frames. These sequence numbers run from 0 up to and including MAX_SEQ, which is defined in each protocol needing it.
- A **packet** is the unit of information exchanged between the network layer and the data link layer on the same machine, or between network layer peers.
- In our model it always contains **MAX_PKT** bytes, but more realistically it would be of variable length.

- A **frame** is composed of four fields: **kind, seq, ack,** and **info**, the first three of which contain control information and the last of which may contain actual data to be transferred.

- These control fields are collectively called the **frame header**.

- The **kind** field tells whether there are any data in the frame.

- The procedures **to_network_layer** and **from_network_layer** are used by the data link layer to pass packets to the network layer and accept packets from the network layer, respectively.

- Note that **from_physical_layer** and **to_physical_layer** pass frames between the data link layer and physical layer.

- In most of the protocols, we assume that the channel is unreliable and loses entire frames upon occasion.

- To be able to recover from such calamities, the sending data link layer must start an **internal timer** or **clock** whenever it sends a frame.

- If no reply has been received within a certain predetermined time interval, the clock times out and the data link layer receives an interrupt signal.

- **Three different protocols** have been proposed to demonstrate the same process on different channels.

1. **An Unrestricted Simplex Protocol.**
2. **A Simplex Stop-and-Wait Protocol.**
3. **A Simplex Protocol for a Noisy Channel.**

3.3.1. An Unrestricted Simplex Protocol:

- Here, Data are transmitted in one direction only.
- Both the transmitting and receiving network layers are always ready.
- Processing time can be ignored.
- Infinite buffer space is available.
- And best of all, the communication channel between the data link layers never damages or loses frames.

This thoroughly unrealistic protocol, which we will nickname "utopia," is shown in the following slide.

- The protocol consists of two distinct procedures, a **sender** and a **receiver**.

- The sender runs in the data link layer of the source machine, and the receiver runs in the data link layer of the destination machine.

- **No sequence numbers** or **acknowledgements** are used here, so MAX_SEQ is not needed.

- The only event type possible is **frame_arrival** (i.e., the arrival of an undamaged frame).

- The sender is in an infinite while loop just pumping data out onto the line as fast as it can.

- The body of the loop consists of **three actions**:

1. Go fetch a packet from the network layer,
2. Construct frame using the variables,
3. Send the frame on its way.

Only the info field of the frame is used by this protocol, because the other fields have to do with error and flow control and there are no errors or flow control restrictions here.

- The receiver is equally simple. Initially, it waits for something to happen, the only possibility being the arrival of an undamaged frame.

- Eventually, the frame arrives and the procedure **wait_for_event** returns, with event set to **frame_arrival.**

3.3.2. A Simplex Stop-and-Wait Protocol.

- Now we will drop the most unrealistic restriction used in protocol 1: the ability of the receiving network layer to process incoming data **infinitely quickly** .

- The communication channel is still assumed to be **error free** however, and the data traffic is still simplex.

- The main problem we have to deal here is " **how to prevent the sender from flooding the receiver with data faster than the latter is able to process them**".
- If the receiver requires a time ***b*** to execute from_physical_layer plus to_network_layer, the sender must transmit at an average rate less than one frame per time ***t***.

- A solution to this dilemma is to have the receiver provide **feedback** to the sender.

- After having passed a packet to its network layer, the receiver sends a little **dummy frame** back to the sender which, in effect, gives the sender permission to transmit the next frame.

- After having sent a frame, the sender is required by the protocol to bide its time until the little dummy (i.e., acknowledgement) frame arrives.
- Using feedback from the receiver to let the sender know when it may send more data is an example of the flow control mentioned earlier.

- Protocols in which the sender sends one frame and then waits for an acknowledgement before proceeding are called **Stop-and-wait**.

- Following slide gives an example of a **simplex stop-and-wait protocol**.

- Unlike in protocol 1, the sender must wait until an acknowledgement frame arrives before looping back and fetching the next packet from the network layer.

- The sending data link layer need not even inspect the incoming frame: there is only one possibility. The incoming frame is always an **acknowledgement.**

- The only difference between receiver1 and receiver2 is that after delivering a packet to the network layer, receiver2 sends an acknowledgement frame back to the sender before entering the wait loop again.

- Because only the arrival of the frame back at the sender is important, not its contents, the receiver need not put any particular information in it.

- Protocols in which the sender waits for a positive acknowledgement before advancing to the next data item are often called **PAR(Positive Acknowledgement with Retransmission)** or **ARQ (Automatic Repeat Request)**.

- Like protocol 2, this one also transmits data only in one direction.

3.4. Sliding Window Protocols:

- In the previous protocols, data frames were transmitted in one direction only. In most practical situations, there is a need for transmitting data in both directions.

- One way of achieving full-duplex data transmission is to have two separate communication channels and use each one for simplex data traffic (in different directions).

- If this is done, we have two separate physical circuits, each with a "forward" channel (for data) and a "reverse" channel (for acknowledgements).

- In both cases the bandwidth of the reverse channel is almost entirely **wasted**.
- In effect, the user is paying for two circuits but using only the capacity of one.
- A better idea is to use the same circuit for **data in both directions**.

- Although interleaving data and control frames on the same circuit is an improvement over having two separate physical circuits, yet another improvement is possible.

- **When a data frame arrives, instead of immediately sending a separate control frame, the receiver restrains itself and waits until the network layer passes it the next packet**(this is possible only when both parties mutually transfer data).

- The acknowledgement is attached to the outgoing data frame (using the ack field in the frame header).

- In effect, the acknowledgement gets a free ride on the next outgoing data frame.
- The technique of temporarily delaying outgoing acknowledgements so that they can be hooked onto the next outgoing data frame is known as **piggybacking**.

- The principal advantage of using piggybacking over having distinct acknowledgement frames is a ***better use of the available channel bandwidth***.

- Other advantage is ***ack*** **field** in the frame header costs only a few bits, whereas a separate frame would need a header, the acknowledgement, and a checksum.

- However, piggybacking introduces a complication not present with separate acknowledgements.

✓How long should the data link layer wait for a packet onto which to piggyback the acknowledgement?

✓ If the data link layer waits longer than the sender's timeout period, what happens??? **The frame will be retransmitted.**

- A solution to the above problem is: **waiting a fixed number of milliseconds**.

- If a new packet arrives quickly, the acknowledgement is piggybacked onto it otherwise, if no new packet has arrived by the end of this time period, the data link layer just sends a separate acknowledgement frame.

- Three protocols have been proposed to overcome the above mentioned problems.
- These three protocols are bidirectional protocols that belong to a class called **sliding window** protocols.
- In all sliding window protocols, each outbound frame contains a sequence number, ranging from 0 up to some maximum. The maximum is usually 2 *N-1.*

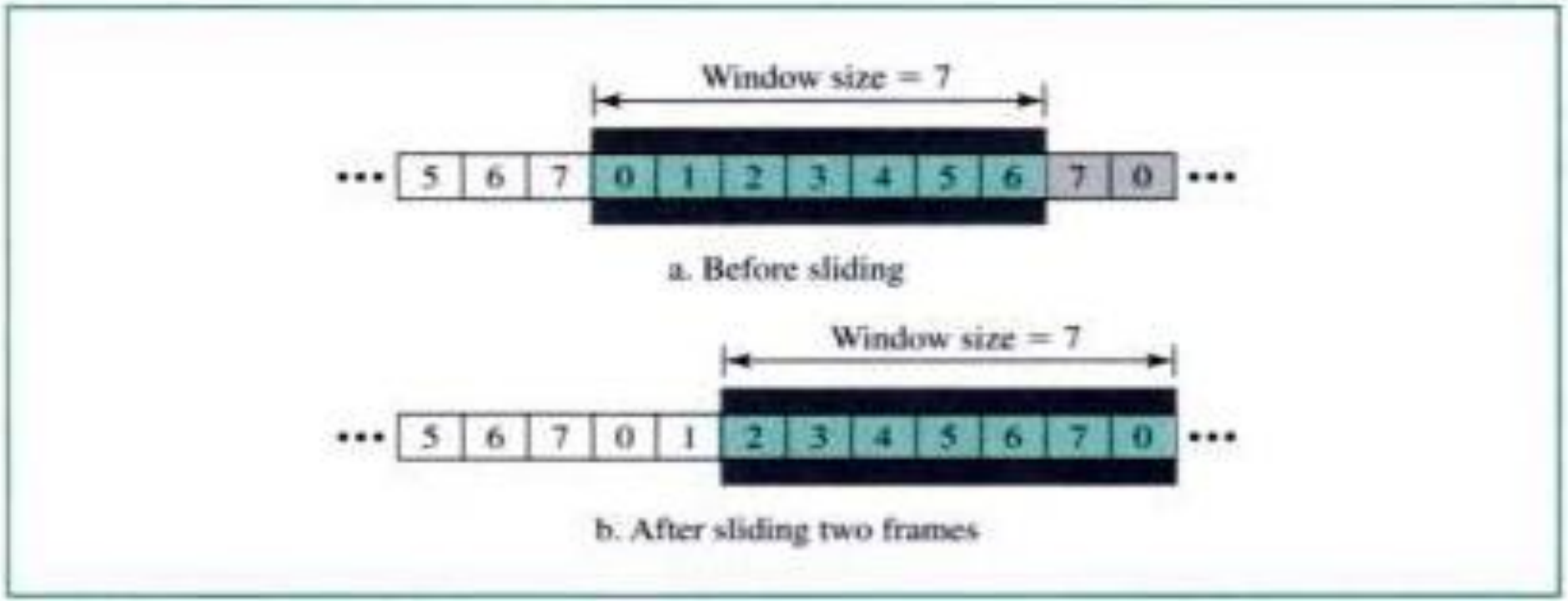
- The essence of all sliding window protocols is that at any instant of time, the sender maintains a set of sequence numbers corresponding to frames it is permitted to send.
- These frames are said to fall within the **sending (sender) window** .
- Similarly, the receiver also maintains a **receiving (receiver) window** corresponding to the set of frames it is permitted to accept.

- In **sliding window** method of flow control, the sender can transmit several frames before needing an acknowledgement.

- The receiver acknowledges only some of the frames, using a single ACK to conform the receipt of multiple data frames.

- The sliding window refers to imaginary boxes at both the sender and receiver.

Sliding Window



- To keep track of which frame has been transmitted and received , frames are numbered ***modulo- n*** , which means they are numbered from 0 to n-1.

- For example if n=8, the frames are numbered 0,1,2,3,4,5,6,7,0,1,2,3,4,5,6,7,0,1…..

- The size of the window is n-1.

- When the receiver sends an ACK, it includes the number of the next frame it expects to receive. For example if the ending frame is 4, the receiver send an ACK containing the number 5.

- When the sender sees an ACK with the number 5, it knows that all frames up through number 4 have been received.

- The three protocols are :

1. **A One-Bit Sliding Window Protocol.**
2. **A Protocol Using Go Back N.**
3. **A Protocol Using Selective Repeat.**

A One-Bit Sliding Window Protocol:

- Before tackling the general case, let us first examine a **sliding window protocol with a maximum window size of 1**.

- Such a protocol uses stop-and-wait since the sender transmits a frame and waits for its acknowledgement before sending the next one.
- Below protocol depicts the same(**One-Bit Sliding Window Protocol).**