Link Layer Protocols

Lecture Notes 3

BASIC IDEA OF ERROR CONTROL

How to make sure that each frame is eventually delivered to the destination (reliable service)

- Reliability through acknowledgement (ACKS) and sometimes negative acknowledgements (NACKS). *Note: NACKS are usually a problem and generally not used.*
- Basic Idea
  - Sender sends a packet and knows that the frame is delivered when it gets a positive acknowledgement.
    - Sender sends, resends if receives a NACK.
    - Receiver sends ACK when the checksum is correct, NACK when incorrect.
  - Problem? What if the whole packet gets lost?
    - Solution: timeout, resend
  - Problem? What if the acknowledgement is lost.
  - Problem? What if the sender has no more data to send.

  Need to do more: sequence numbering, ordering of exactly how to handle timeout and receipt of acks, etc.

Logical Format of an IP Packet

Version 4 bits	IHL 4 bits	Service Type 8 bits	Total length 16 bits
Identification 16 bits		Flags 3 bits	Fragment offset 13 bits
Time to Live 8 bits	Protocol 8 bits	Header Checksum 16 bits	
Source IP Address 32 bits			
Destination IP Address 32 bits			
IP Options if used plus padding to 4 bytes Variable length multiples of 4 bytes			
Encapsulated Data Variable length, integral number of bytes			

UDP Logical Format

source port 16 bits	destination port 16 bits
UDP length (in bytes) 16 bits	checksum 16 bits
Data, variable length, multiple bytes	

TCP Logical Format

<table>
<tr><td colspan="3">Source port
16 bits</td><td>Destination port
16 bits</td></tr>
<tr><td colspan="4">sequence number
32 bits</td></tr>
<tr><td colspan="4">acknowledgement number
32 bits</td></tr>
<tr><td>Hlen
4 bits</td><td>Reserved
4 bits</td><td>8-bit control
8 bits</td><td>window size
16 bits</td></tr>
<tr><td colspan="3">checksum
16 bits</td><td>Urgent pointer
16 bits</td></tr>
<tr><td colspan="4">Options plus pad, variable length</td></tr>
<tr><td colspan="4">Data, variable length</td></tr>
</table>

Flow control and Congestion Control

Flow control

- To avoid swamping the slow receiver.
- Use a feedback mechanism to make sure that the sender knows the status of the receiver.
  - "send me N frames and no more until I tell you to continue"
- Based typically on the notion of a flow

Congestion Control

- Avoiding congestion in the network
- Credit and token schemes
- Access control

ARQ PROTOCOLS

- Automatic Repeat Request
  - Basis for peer-to-peer protocols that provide reliability
  - Used on link layers or end to end
  - Generally point to point (Transmitter and Receiver)
  - Used for noisy or lossy channels where errors occur
- Terminology
  - Frame: basic transmission block consisting of header, user information, and trailer
  - Information frames and control frames
  - Acknowledgements: a reply from the peer that indicates status
    - Positive ack: I got it
    - Negative ack: I did not get …….
  - Piggy Back acks: acks sent as part of an information frame rather than isolated
- Sequencing: numbering frames to keep them in order and keep track of which one is received
- Timeouts: used to determine when to retransmit
- Retransmissions: resending a frame potentially lost
- Window: keeping track of a set of frames for some purpose

Simplest protocol
P0
(not very good)

- Sender
  - get a packet from the upper layer
  - Determine the header
  - Add frame header, send the frame via the lower layer.
- Receiver
  - get a frame from the lower layer
  - Analyze response based on the header
  - deliver the packet to the upper layer.

Why is this protocol not very good?

P1: First attempt at simple stop and wait

Assumption: no lost packets

- Sender
  - get a packet from the upper layer
  - Add frame header, send the frame to the physical layer.
  - Wait for ACK, if correct send next packet else resend current packet
- Receiver
  - get a frame from the upper layer
  - Check header. If correct, send ACK and deliver packet to upper layer
  - If incorrect, send a NACK

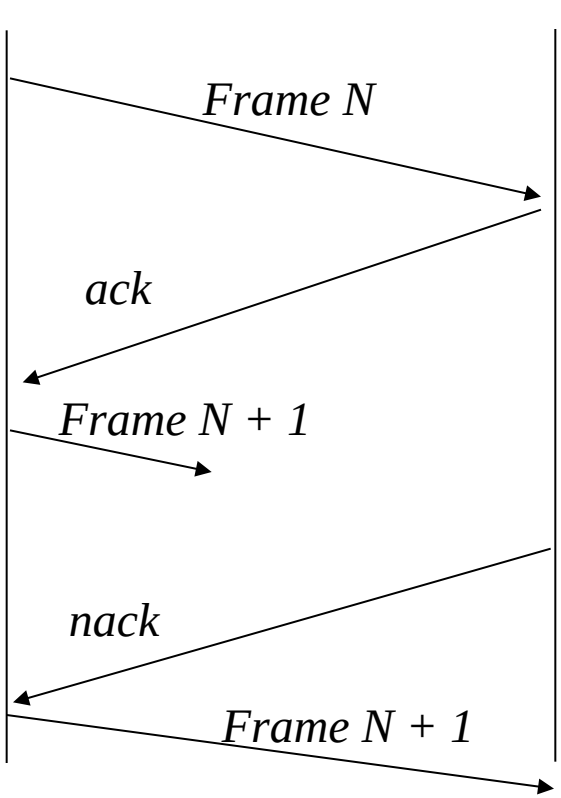
What is wrong with this protocol?

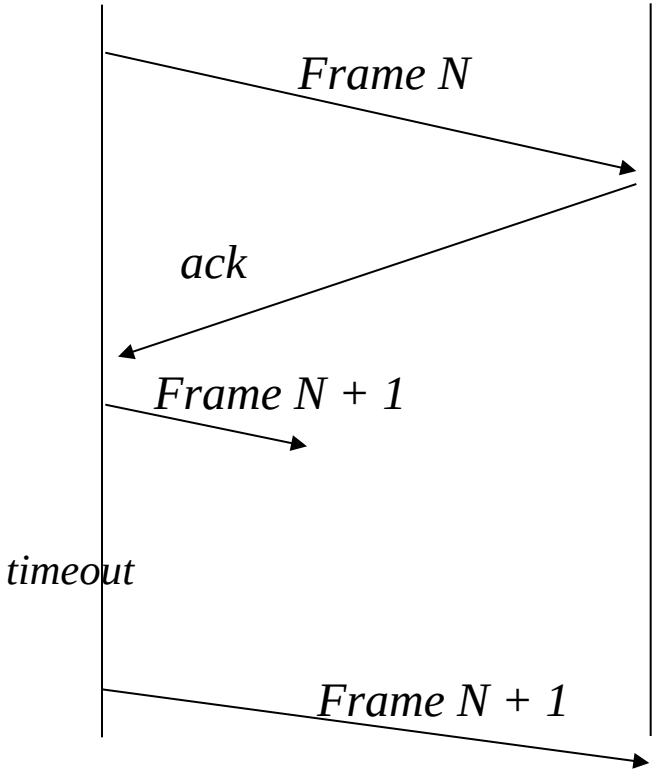
Developing a better protocol

Problem: frame may get lost, ack may get lost, how long do you wait

1. send frame
   wait for ack or nack;
2. on timeouts or nacks retransmit (use one or the other)

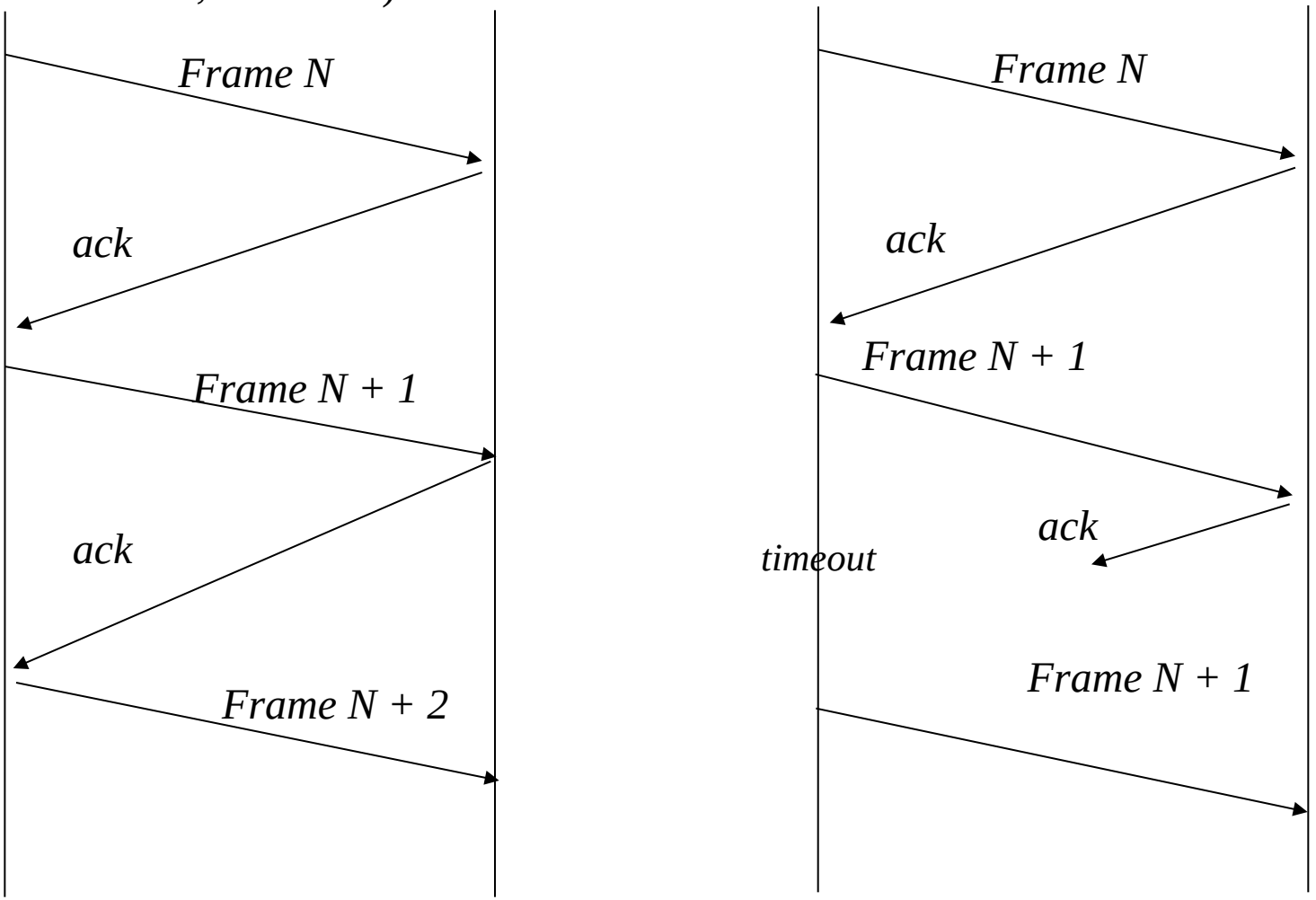
Return ack (or nack)





Developing a better protocol (continued)

3. Problem: transmitter behaves the same but *receiver* cannot distinguish between retransmitted and new frame without some distinguishing characteristic. (Note: using timeouts, not nacks)



How can the receiver distinguish between getting N+2 and getting N+1 again

Developing a better protocol (continued)

- Add a 1 bit sequence # to the frame
  - Frame N: seq num 0
  - Frame N+1: seq num 1
  - Frame N+2: seq num 0

- This is often called the alternating bit protocol
- Uses only positive acks (no nacks)
- We need to similarly distinguish acks (ie ACK0 or ACK1) due to delayed acks otherwise we can have problems also.
- Note: conventionally, ACK1 means I am expecting frame sn = 1 and I got frame sn = 0.

P2: stop and wait + timeout + sequence numbers

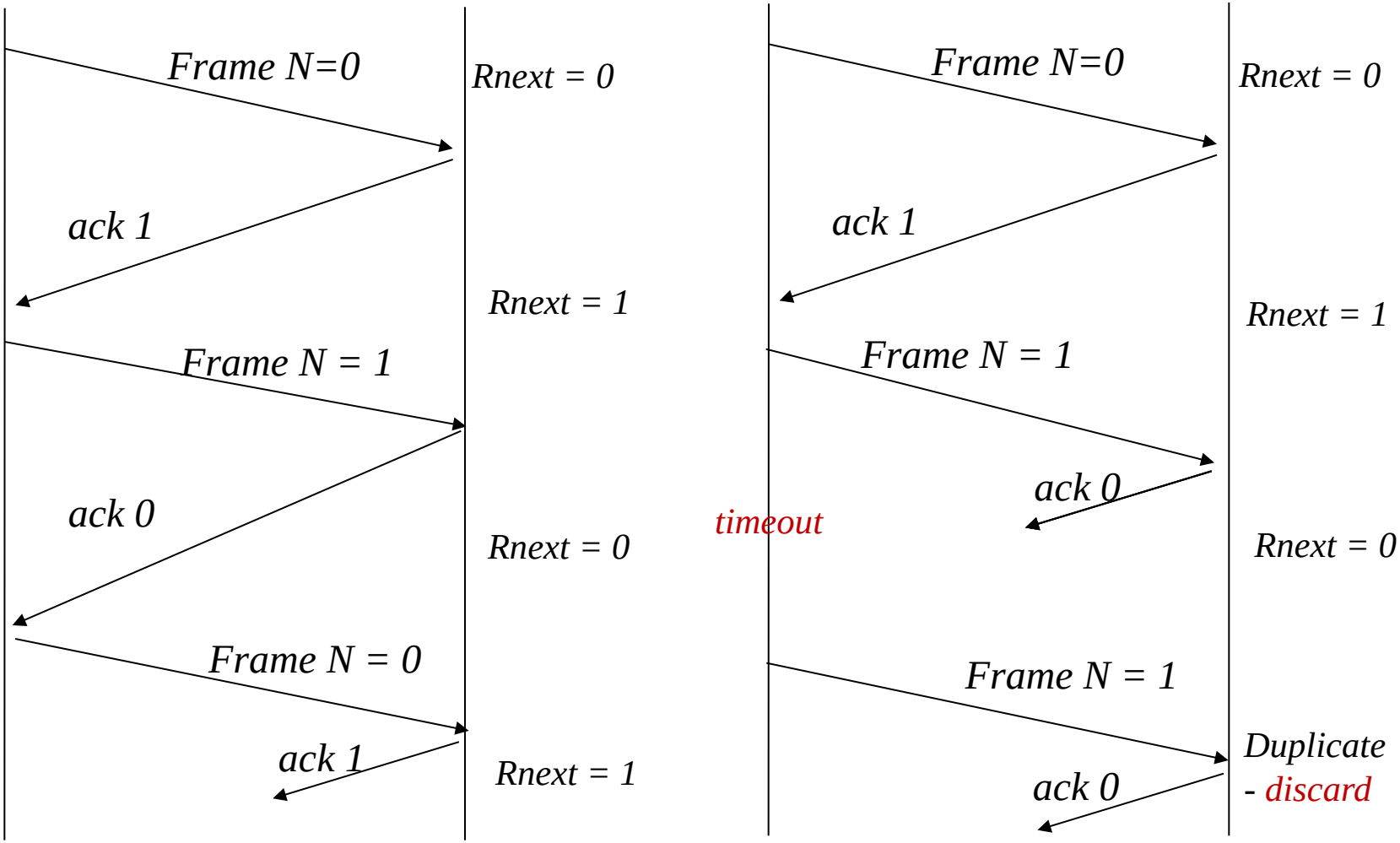
- **Sender**
  - Send frame (sn)
  - Wait for ACK or timeout
    - if timeout, resend frame (sn)
    - If correct ack, reset timeout and send new frame (sn + 1)
    - If incorrect ack, ignore and wait for timeout
- **Receiver**
  - Receive a frame
  - if (sn = expected sn)
    - Deliver packet.
    - Send ACK (expected + 1)
  - Else
    - send ACK (expected)

Starting condition: sender initially sends a sequence number (sn) 0; Receiver initially expects to receive sequence number 0

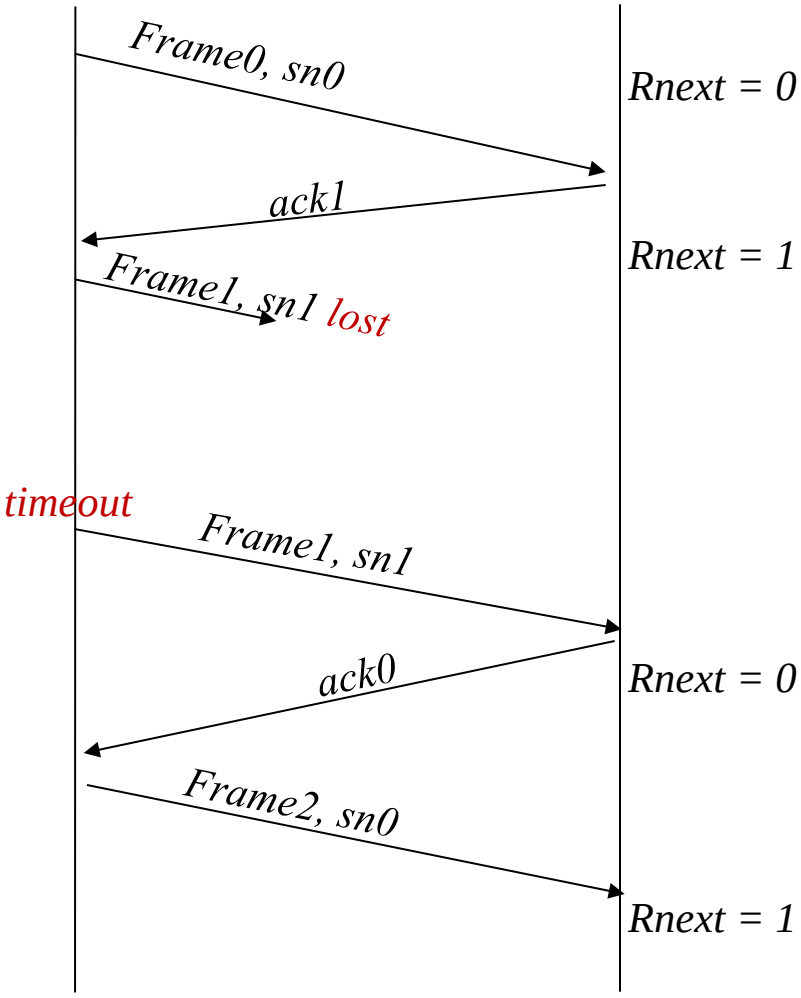
Receiver ack number indicates the next sequence number it expects to receive

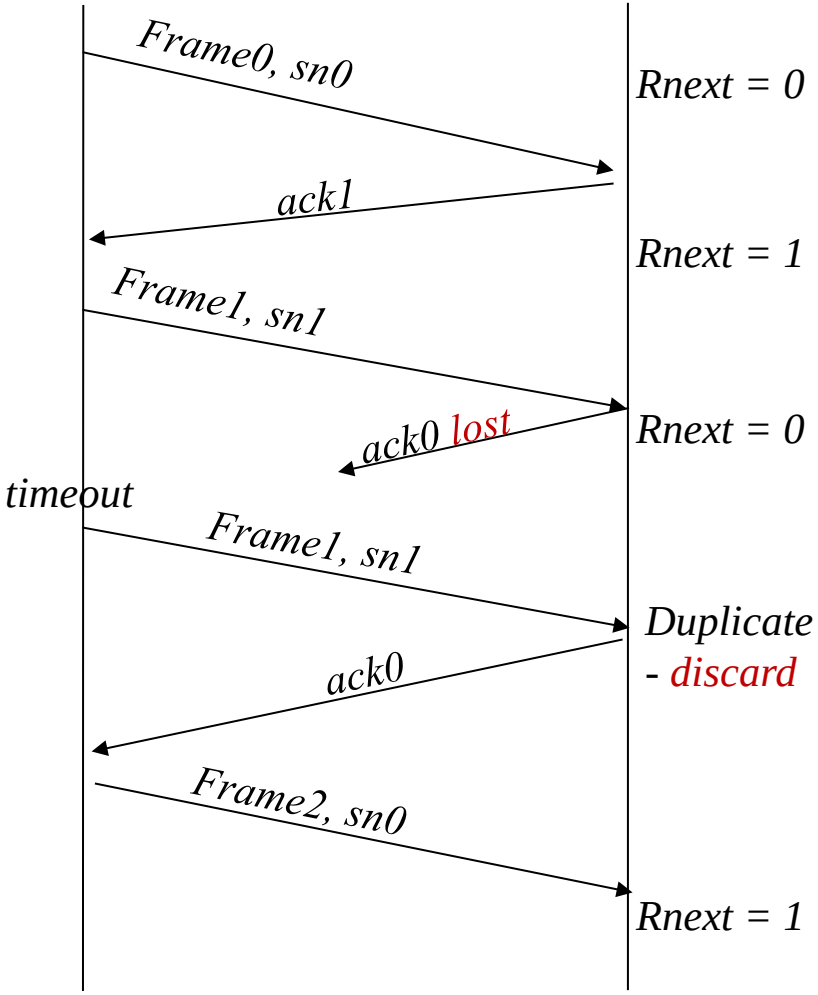
P2: stop and wait + timeout + sequence numbers

The receiver can distinguish between N + 2 and N + 1

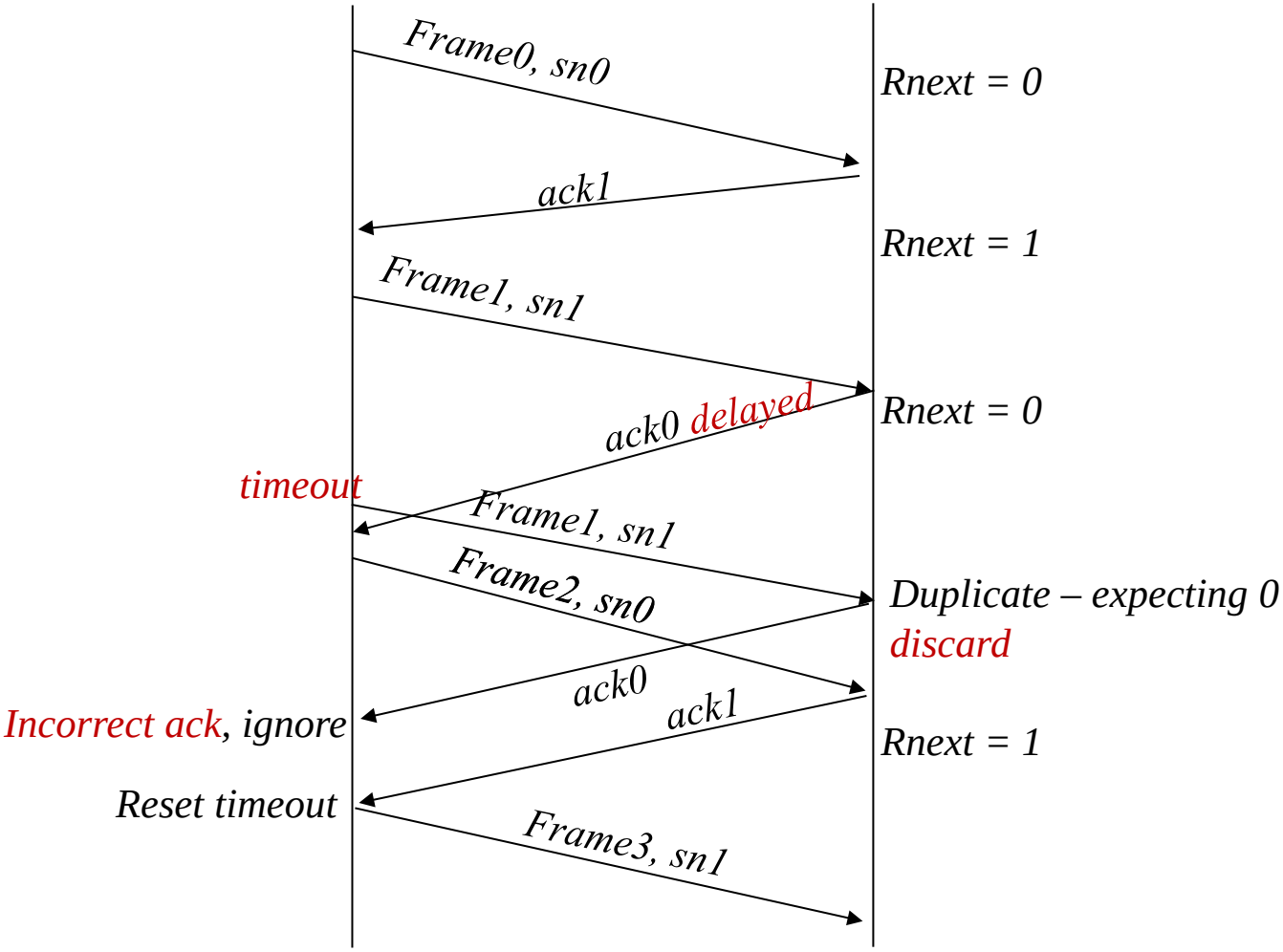


Alternating Bit Solution
Another example of handling losses



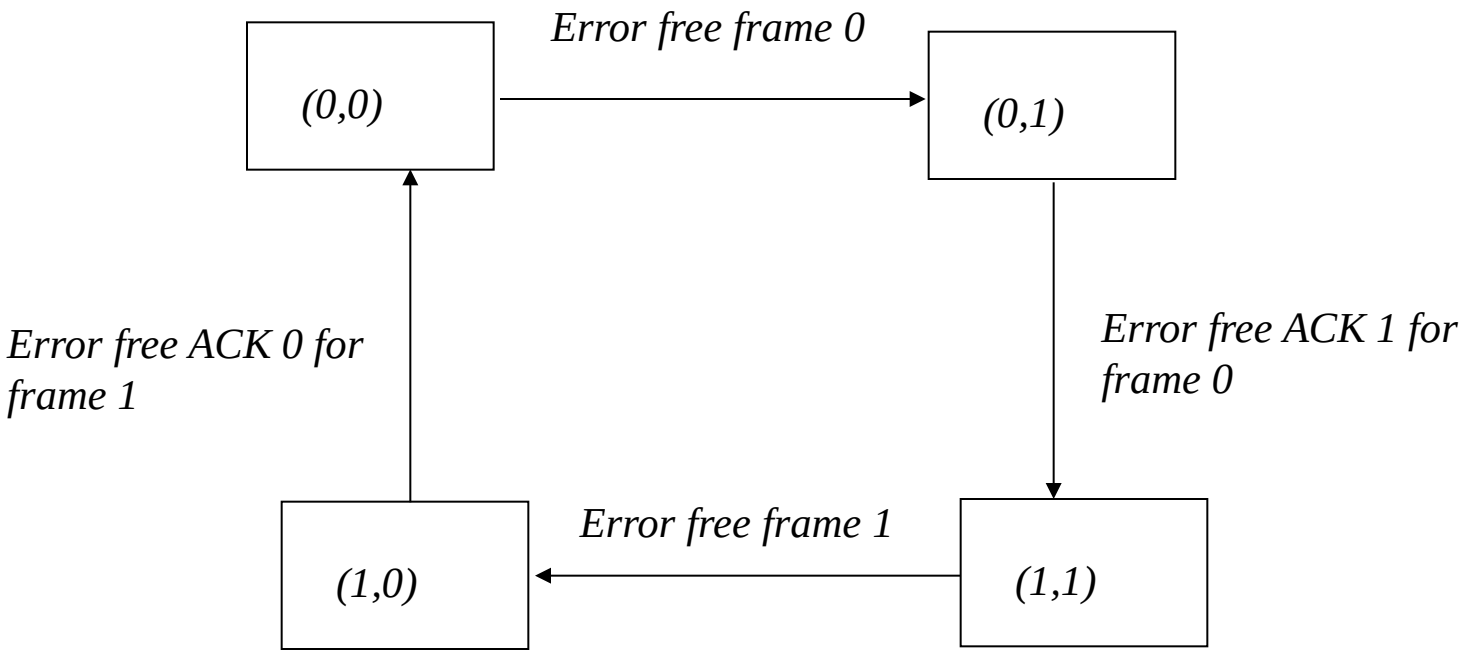


Alternating Bit Solution
An example of delayed Ack



Simplistically analyzing correctness using system state

Global state (S_{last}. R_{next})



Claim: correct and orderly delivery of frames. But is that enough?

P2 improved: stop and wait + timeout + sequence numbers + fast response to incorrect ack

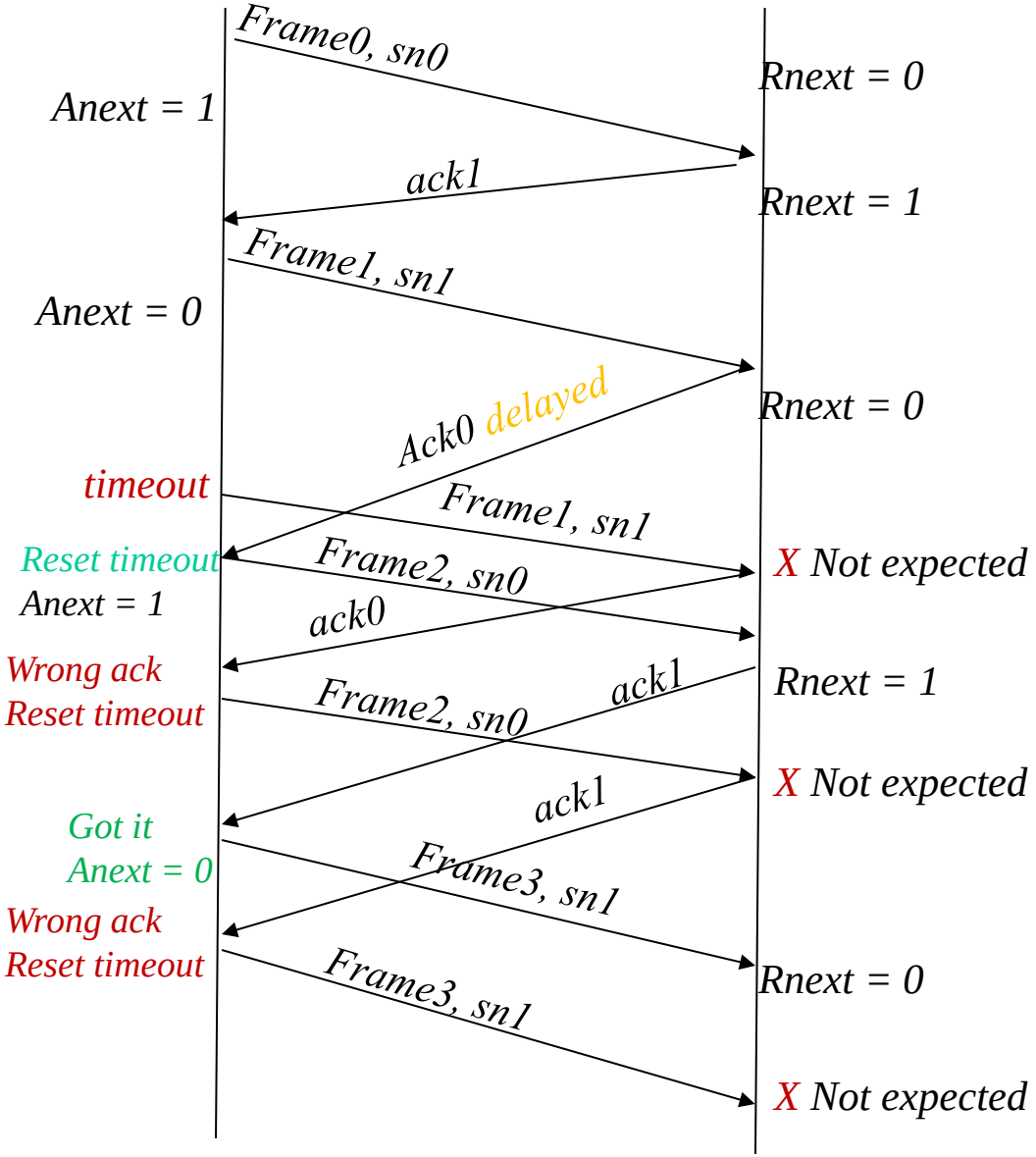
- **Sender**
  - Send frame (seq)
  - Wait for ACK or timeout
    - if timeout, resend frame (seq)
    - If correct ack, reset timeout and send new frame (seq + 1)
    - If "incorrect ack," reset timeout and resend frame

- **Receiver**
  - Receive a frame
  - if (seq = expected)
    - Deliver packet.
    - Send ACK (previous + 1)
  - Else
    - send ACK (previous)

If incorrect ack rather than "ignore and wait for timeout"

we immediately reset timeout rather than ignore the ack.

P2 Improved is actually flawed! Sorcerer's Apprentice Problem



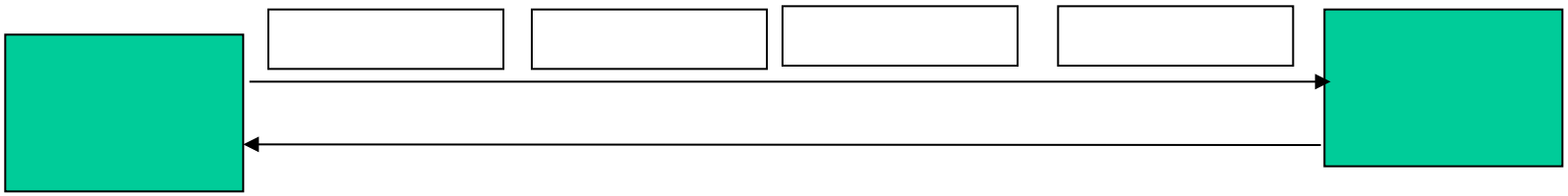
When a wrong ack comes, the ack should be ignored, nothing should be sent and the timeout should not be reset, otherwise will cause duplicate packets to be sent continuously

Protocol 3: Sliding window protocol (improved over P2)

- What is bad about protocol 2?
- How to improve?
  - Allow pipelining, more outstanding frames.
  - Maintain a window of outstanding frames at the sender (sender window)
    - All packets that are sent but not ACKed.
  - Maintaining the sender window (some details):
    - increase the upper bound of the window when a new packet is given from upper layer.
    - Increase the lower bound when an ACK of the lower bound frame arrives.
  - Need to decide how / if to use window at receiver

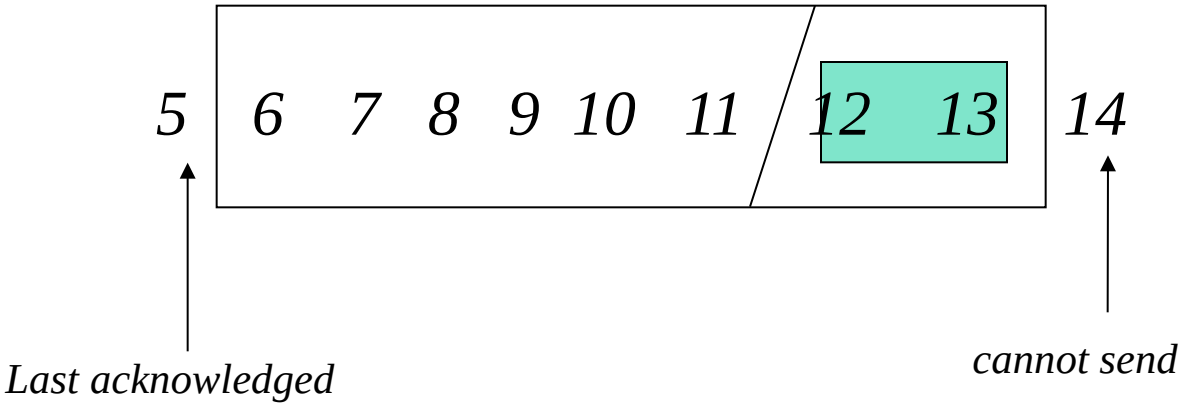
Bandwidth and delay and pipelining

- Suppose round trip time between sender and receiver is 4 seconds (*delay*). Suppose channel is 2000 bits per second (*bandwidth*). Then, to keep the channel "full" we can send 8000 bits (*delay* × *bandwidth*) until a response arrives from the receiver.
- Suppose packets are 1000 bits long. Then, we can send 8 packets until a response arrives.



A sliding window

Window of size 8: sent and unacknowledged (outstanding), or possible to send frames



Sequence numbering frames

- 0, 1, 2, …, max-sequence-number
- With 3 bits have 0, 1, 2, …, 7
- Let R_{window} be size of receive window
- Let S_{window} be size of send window
- Let N be the size of the sequence space

- Claim: $S_{window} + R_{window} \leq N$

- In go back N, the receive window is size 1 thus the max size of the send window with 3 bits is 7.

Protocol: Go back N sliding window.

- Allow the sender to have multiple (N > 1) outstanding frames.
- Sender window = N (>1)
- Receiver window = 1.

- Sender: maintain the sender window (buffering all outstanding frames), maintain a timeout for all outstanding frames.
- Receiver: Similar to protocol Stop and Wait (expected sequence number is longer). When expected frame is received ACK by sending next frame expected. When an out of order frame is received discard the frame and ACK the last frame received.

- What happens when an transmission error occurs?
- When timer expires, resend all buffered frames GO BACK N to resend. Note: need to maintain only a single timer

- Example: Frames 0,1, 2, 3, 4, 5, 6, 7, 8, 9 sent.
- ack 1, ack 2, ack 3, no ack 4 (acknowledging frame 3) received
- Frame 3 was lost, so we need to resend 3. But since the receiver has only a window of 1 we resend 3 onwards, even if they were being received correctly

Sending 8 or 7 frames with 3 bit sequence space

- Suppose 8 frames were allowed in sender window.
  - Sender sends 0, 1, …, 7.
  - Receiver acks 0, 1, …, 7.
  - All acks are lost
  - Sender retransmits 0.
  - Receiver is expecting new frame 0 (really 8).
  - How can it distinguish from retransmitted frame 0?

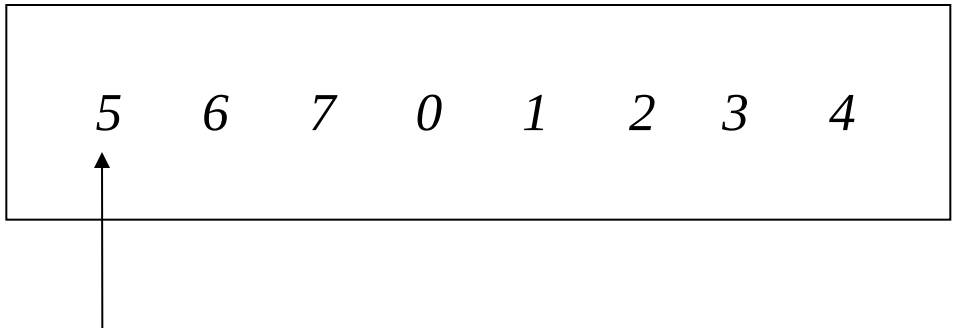
- Suppose 7 frames were allowed in sender window
  - Sender sends 0, 1, …, 6.
  - Receiver acks 0, 1, …, 6.
  - All acks are lost.
  - Sender retransmits 0.
  - Receiver is expecting 7 so it knows that 0 is an old frame. Similarly with the frames though 6. When it gets 7, and is expecting 0, then the sender window must have moved to 1, and thus the next 0 it receives will be frame 8.

- How can we further improve the performance of the Go back N sliding window protocol?

  - When no errors, go back n can achieve the best communication performance (fully exploit the pipeline communication).
  - When an error occurs, the go back n protocol needs to resend everything.
    - How can we do better than that?

- The receiver now maintains a window of packets that it is willing to keep, even though they might be out of order.
- Maintaining the receiver window:
  - discard frames outside the window.
  - When receiving the frame whose sequence number = lower bound, ACK and advance the window (the ACK sequence number may jump)
  - Can use NAK to speedup retransmission.

Receive Window of size 8



Protocol: Selective repeat sliding window.

- Allow the sender to have multiple (N > 1) outstanding frames.
- Allow the receiver to receive multiple (M>1) frames.
- Sender winder = N (>1)
- receiver winder = M (> 1).

- Sender: maintain the sender window (buffering all outstanding frames), maintain timeout for all outstanding frames. Keeps track separately of which Selective ACKs have been received and moves the window as appropriate.
- Receiver: maintain the receiver window (buffer out of order packets for in order delivery), sends selective ACKs for packets received (including those below the window).

- What happens when a timeout occurs?
  - resend only the selected packet.

– Try this example: Let one way delay equal to the time for sending 2 packets. Timeout time = round trip time + time for 2 packets. Let the sender sends 10 packets and assume the fourth packet gets lost.

  - Using go-back-N with sender window size = 8.
  - Using selective repeat with sender window size and receiver window size = 8

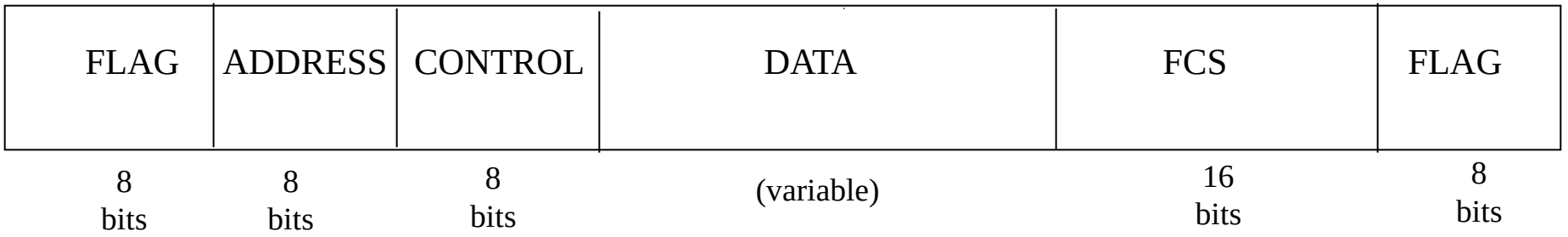
Synchronization –bit stuffing

- Distinctive pattern is used to identify start and end of frame. For example
  - 01111110 identifies start and end of frames
  - 11111111 identifies an idle line
- Once the flag is discovered, rest is frame content until closing flag
  - Problem – Content may have flag pattern by chance
  - Solution – Bit stuffing: after the opening flag, enable circuitry that inserts a 0 after any 5 consecutive 1s in the data. Receiver does the opposite. Not done for the closing flag.
  - Note that a 0 is “stuffed” into the stream

High Level Data Link Control (HDLC)

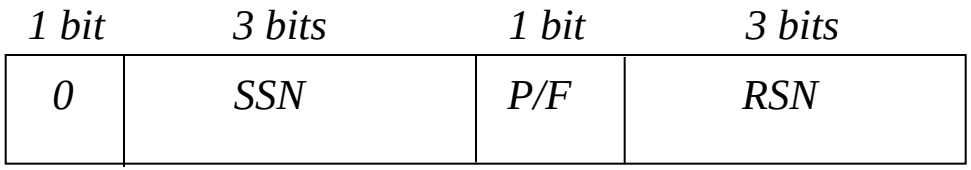
- The prototype for many other link layer protocols
  - Makes a link appear to be reliable (ARQ)
  - Implements basic flow control
    - Sets a limit on the number of outstanding frames without an ACK
    - Receiver can indicate its buffers are full and request sender to stop sending
  - Uses bit oriented framing
  - Originally designed for multi-drop lines (multipoint) with a supervising entity exchanging frames with a number of secondary entities: addressing is thus very simple in HDLC with an 8 bit field in the basic protocol
  - Supports several data transfer modes of which the most important are:
    - Normal Response Mode (NMR): A primary sends commands to several secondary entities (also called an unbalanced multipoint link)
    - Asynchronous Balanced Mode (ABM): two stations act as peers with each acting the role of primary and secondary station. Information frames are sent in both directions for full-duplex transmission

Logical HDLC frame format

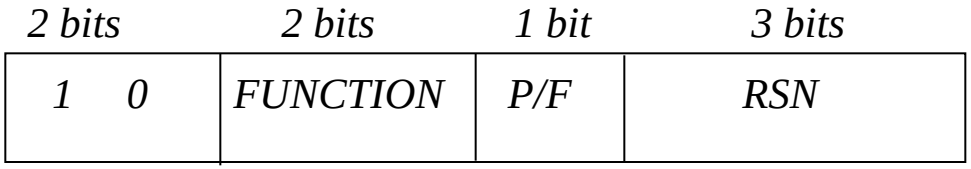


- Flag: the bit pattern 01111110
- Address: used for disambiguating entities on a multi-drop line
- Control: Indicates the type of frame and has other associated information
- Data: the user data of variable length
- FCS: frame check sequence using a CRC code

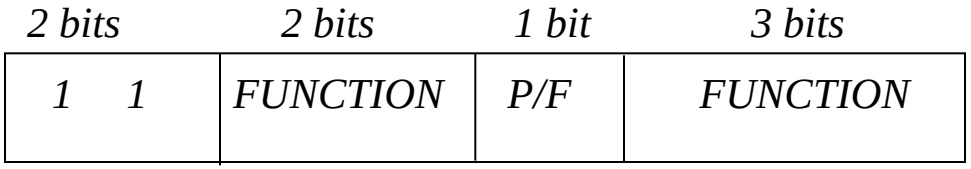
Format of the control fields



Information (I)



Supervisory (S)



Unnumbered (U)

- SSN is send sequence number, RSN is receive sequence number
- P/F is the poll / final bit
- Function bits identify further the control frames

The supervisory frame

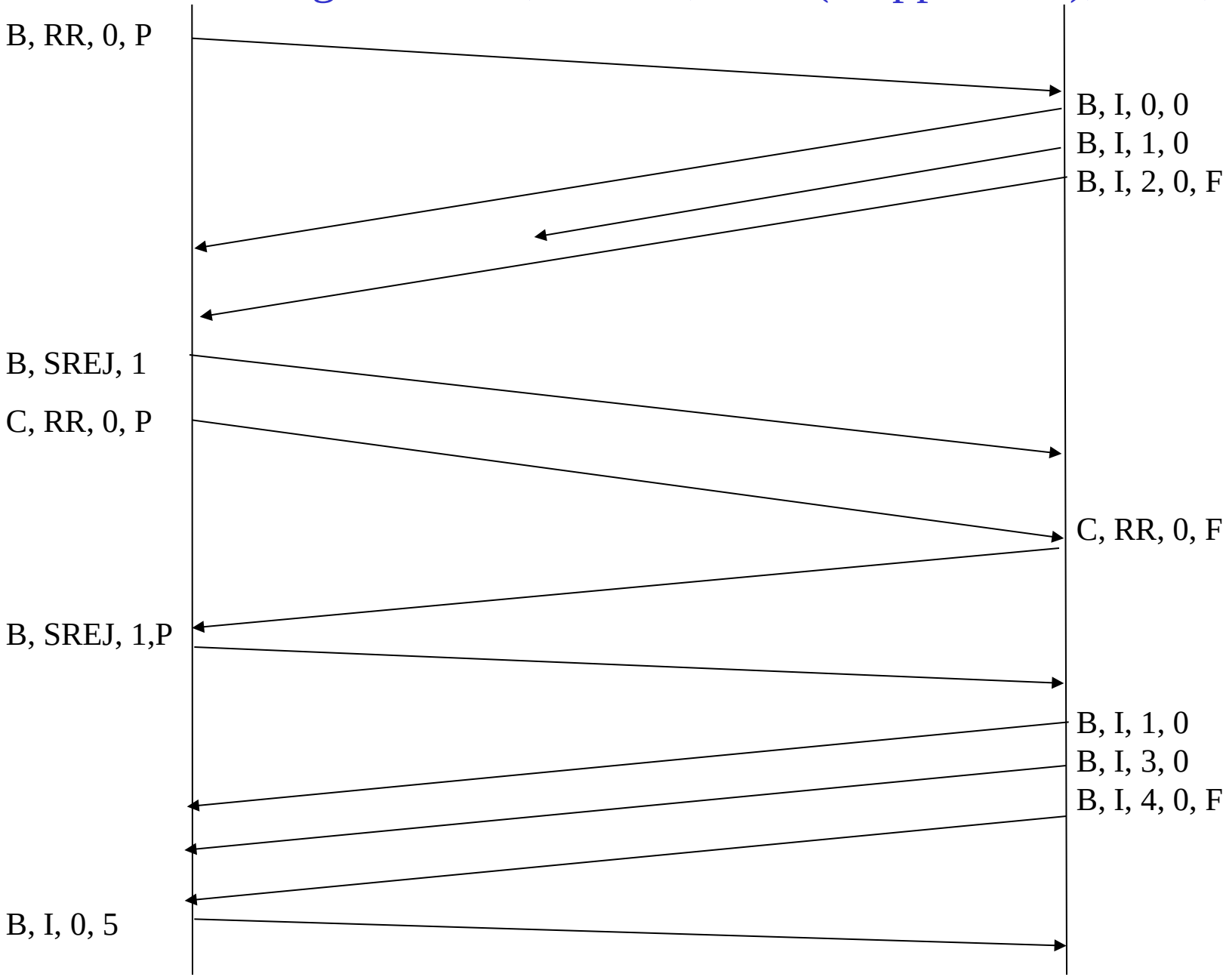
- Receive Ready (RR)
  - Used to acknowledge frames when no information frame is ready for piggybacked acknowledgements
- Reject (REJ)
  - Tells the other entity that an error was detected and to retransmit frames from RSN on (Used in the GO BACK N case)
- Receive not Ready (RNR)
  - Acknowledges all frames less than RSN and tells other entity not to send any more frames because of buffers being full, etc.
- Selective Reject (SREJ)
  - Retransmit the frame with sequence number RSN (Used in the Selective Repeat ARQ case)

Unnumbered frames and P/F bit

- Unnumbered frames
  - SABM: set asynchronous balanced mode
  - SNRM: set normal response mode
  - SABME: set asynchronous balance mode extended
  - DISC: disconnect
  - UA: unnumbered acknowledgement
  - FRMR: frame reject
- The unnumbered frames are used in initialization, to define parameters for the connection, and for disconnecting

- Poll / Final Bit
  - When set and sent by the primary in unbalanced mode, indicates a poll of the specific secondary and requests transmission of frames
  - When set by the secondary it indicates the final frame
  - In balanced mode, it is used for sending an ENQ control message asking the other side to send back its status.

An example exchange: primary A and secondary entities B, C
format of exchange: address, control, SSN (if applicable), RSN, P/F



The point to point protocol (PPP)

- The Point-to-Point protocol is designed to send encapsulated packets such as IP datagrams over serial connections. It should be viewed as a link layer protocol. It is a descendent of HDLC. Note however that PPP does not need to have a separate physical address such as a MAC address, since there is no address ambiguity on a point-to-point link. However, for convenience, PPP does support an address for each end of the link – fortunately or unfortunately these are IP addresses.

- PPP supports multiple protocols besides TCP/IP; supports negotiation for communication parameters of the link; supports error detection (but usually not flow control, error correction, re-sequencing or retransmission); supports compression, authentication, and encryption; and supports the establishment of IP addresses.

- PPP is both a control protocol and a data transport protocol. The control part negotiates various aspects of the connection before sending encapsulated data.

PPP frame format

- [flag] 1 byte
  [addr] 1 byte, not really used, fixed value
  [ctrl] 1 byte, fixed value “unnumbered information”
  [protocol] 2 bytes, default value, smaller can be negotiated.
  [data] variable, depends on the protocol value
  [fcs] 2 bytes, default
  [flag] 1 byte, ending flag

Address usually set to 11111111 indicating that all stations should accept this frame

The control field is generally set to 00000011 indicating the analogue of an HDLC unnumbered frame and is thus used only in the connectionless mode without acks

The protocol field indicates the protocol carried as part of the information field

Examples are Link Control Protocol for setting up, configuring,, testing and terminating the link, Network Control Protocol, IP, etc.

LCP – Link Control Protocol

- LCP is used to establish, maintain and end a PPP session.
- An LCP packet is carried in the data portion of the PPP frame identified by a code value:
- config request, config ack, config nak, config reject, terminate request, terminate ack, (some others …).
- These LCP packets are used to negotiate a series of options such as ACCM (async control character map), the type of authentication protocol to use (CHAP - challenge handshake authentication protocol), MRU etc.
- The format of the LCP packet in the PPP data part is:

[LCP code] 1 byte, code value of the control packet
[ID] 1 byte, used to match requests with responses
[length] 2 bytes, length in bytes of the LCP packet
[LCP data] variable

Example of establishing a session

- Parameters of the link, say home computer to POP are established in each direction separately
  - Home computer establishes from POP to Home Computer
  - POP establishes in other direction
- Suppose a set of 3 options are established, including use of authentication
  - There would likely be 4 message exchanges for each simplex direction establishment with a sequence such as: creq, cnak, creq, cack
  - Each message (containing suggested values of the 3 options) would be on the order of 27 bytes as follows:
  - [flag, addr, crtl, *LCP*: 5] [CREQ, id, length: 4] [*options*: 15] [fcs, flag: 3]
  - In this example, negotiation would involve 8 messages x 27 bytes to establish both links

Authentication Phase

- If an authentication protocol, say CHAP (Challenge-Handshake Authentication Protocol) was proposed, this is done next.
- For example, assume that we are using MD5 keyed hash. Both Home and POP must assume a shared CHAP secret key. Assume the POP wishes to authenticate the home computer. The three-way handshake proceeds as follows, beginning with the POP: challenge, response, acknowledgement (success or failure).
  - POP sends a challenge string of size 60 bytes and a host name of 4 bytes
  - Response includes a 16 bytes MD5 authentication hash.
  - POP responds with an acknowledgement of success and a welcome

Network Control Protocol

- A network control protocol is next used to set up TCP/IP options
- The structure of negotiating this is similar to LCP but the options are different
  - For example, suppose we use the network control protocol IPCP (internet protocol control protocol)
  - Options could additionally be to use Van Jacobson compression and the Home Computer might request an IP address from the POP.

- Once all these options are established, PPP can begin carrying IP packets in its data portion
- *Excluding negotiations, what would the overhead be for transporting IP packets?*