

L5: Building Direct Link Networks III



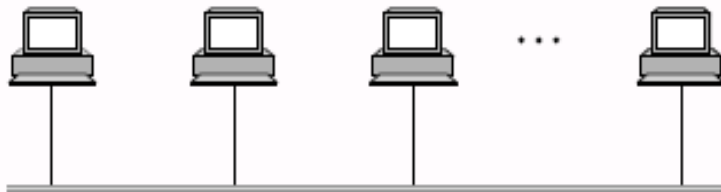
Hui Chen, Ph.D.
Dept. of Engineering & Computer Science
Virginia State University
Petersburg, VA 23806

Acknowledgements

- ❑ Some pictures used in this presentation were obtained from the Internet
- ❑ The instructor used the following references
 - Larry L. Peterson and Bruce S. Davie, Computer Networks: A Systems Approach, 5th Edition, Elsevier, 2011
 - Andrew S. Tanenbaum, Computer Networks, 5th Edition, Prentice-Hall, 2010
 - James F. Kurose and Keith W. Ross, Computer Networking: A Top-Down Approach, 5th Ed., Addison Wesley, 2009
 - Larry L. Peterson's (<http://www.cs.princeton.edu/~llp/>) Computer Networks class web site

Direct Link Networks

- Types of Networks
 - Point-to-point
 - Multiple access



- Encoding
 - Encoding bits onto transmission medium
- Framing
 - Delineating sequence of bits into messages
- Error detection
 - Detecting errors and acting on them
- **Reliable delivery**
 - **Making links appear reliable despite errors**
- Media access control
 - Mediating access to shared link

Reliable Transmission

- ❑ How to make unreliable links appear to be reliable?
- ❑ What to do when a receiver detects that the received frame contains an error?

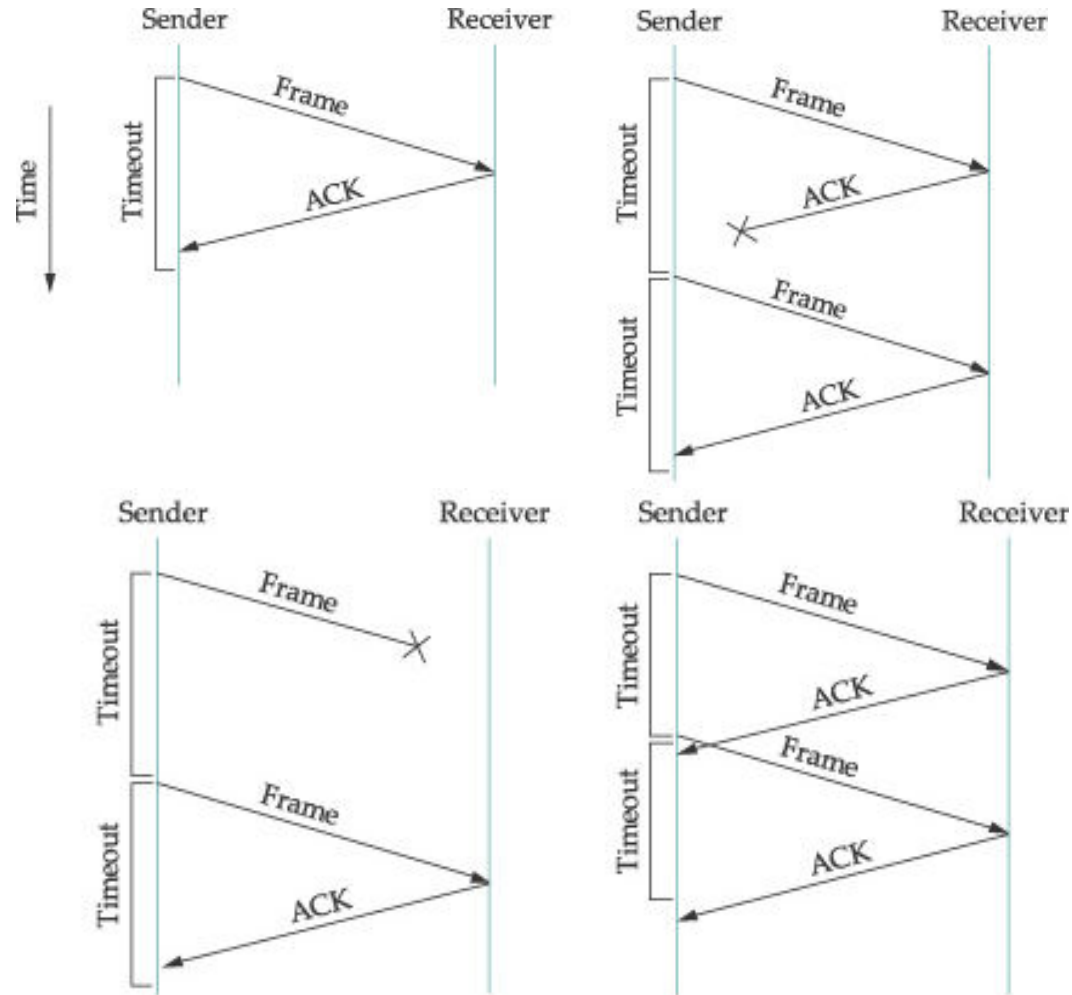
Acknowledgment and Time-Out

- Two fundamental mechanisms to make channels appear to be error-free
 - Acknowledgements (ACK)
 - Time out
- Automatic Repeat Request (ARQ)
 - Stop-and-Wait
 - Sliding Window
- Discuss Stop-and-Wait and Sliding Window protocols in *the context of point-to-point links*

Stop-and-Wait

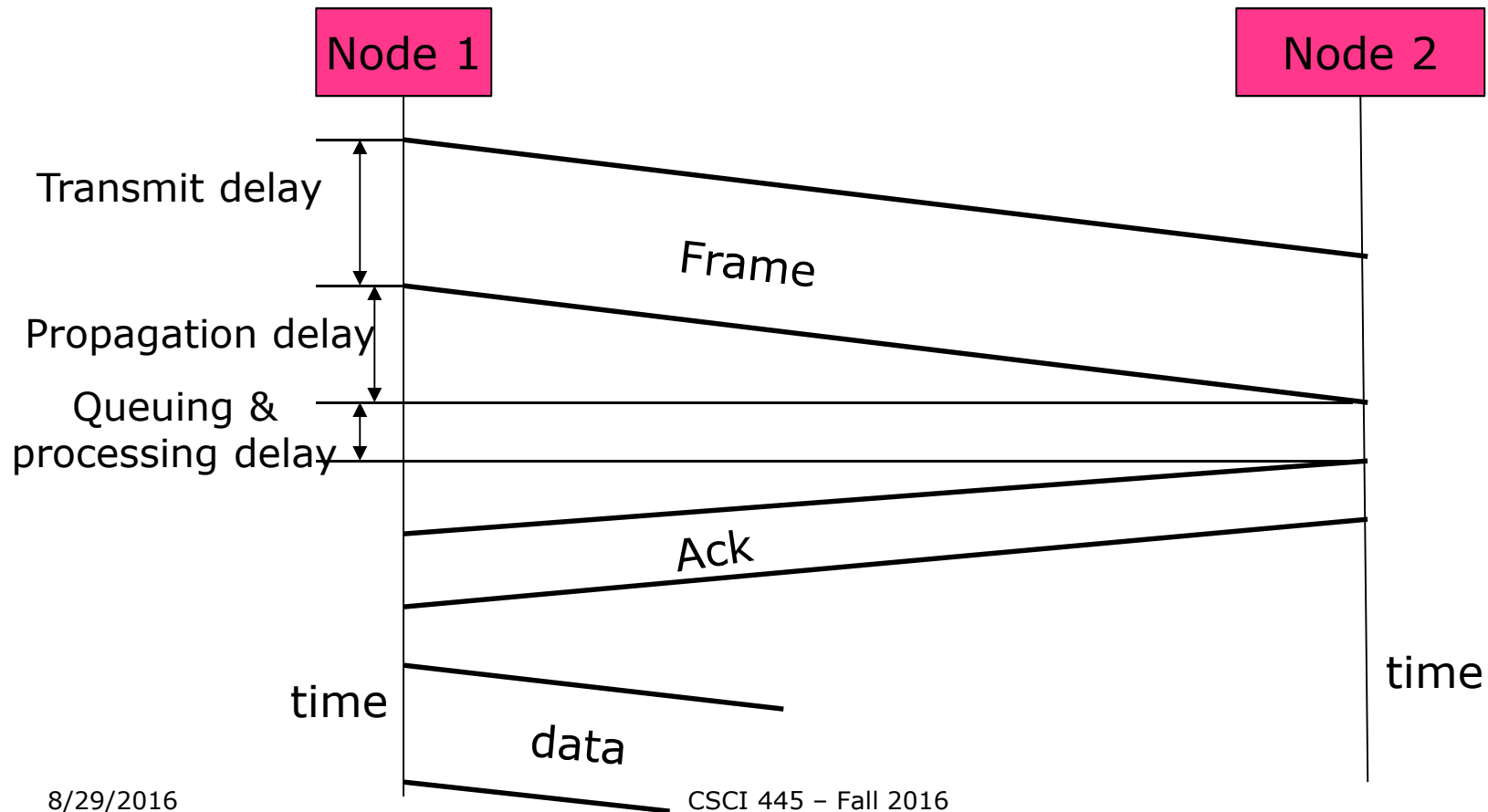
- ❑ Sender transmits a frame
- ❑ Sender *waits* for an acknowledgement before transmitting the next frame
- ❑ If no acknowledgement arrives after a *time-out*, the sender times out and *retransmits* the original frame

Stop-and-Wait



Performance

- Performance analysis for the stop-and-wait protocol with point-to-point links

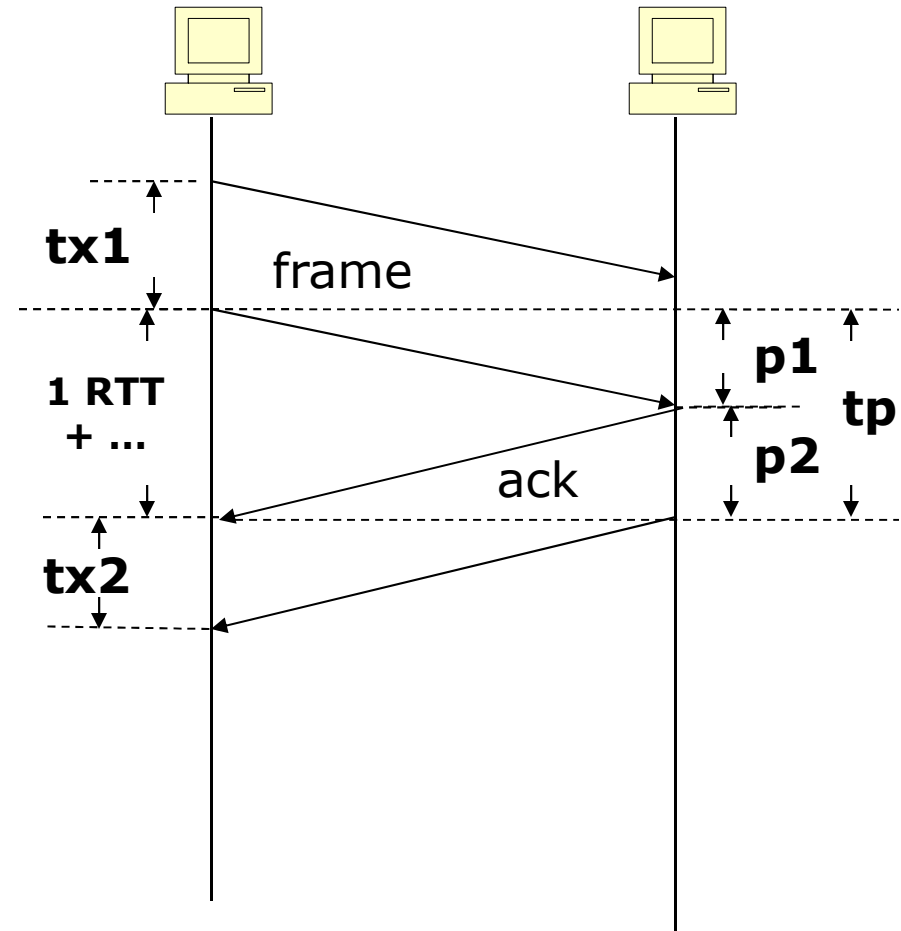


Example

- ❑ Link bandwidth: 10 Gbps
- ❑ RTT = 40 ms
- ❑ Frame size = 1500 bytes
- ❑ Acknowledgement size = 64 bytes
- ❑ Timeout: $2 \times \text{RTT}$
- ❑ Assume processing delay is 0
- ❑ Stop-and-Wait protocol: receiver transmits acknowledge frame upon receiving the data frame
- ❑ Q: what is the **maximum** throughput (effective bandwidth)?

Throughput

- ❑ Q: what is the **maximum** throughput (effective bandwidth)?
- ❑ Note: $tp = p1 + p2 = 1 \text{ RTT}$
- ❑ Transfer time = $tx1 + tx2 + tp$
- ❑ Throughput =
Transfer size/Transfer time
- ❑ Q: Is this a good protocol?



Timeout?

- How long should the receiver wait?

- Timeout: 2 x RTT or more ...

Exercise L5-1

- ❑ Data frame size (data) = 1500 bytes
- ❑ Acknowledgement frame size (ack) = 64 bytes
- ❑ Stop-and-Wait protocol: *receiver is forced to wait 1 RTT before transmitting acknowledgement frame after having received data frame. No additional processing and queueing delay*
- ❑ Draw timeline diagram first, and then compute throughputs and link bandwidth utilization for *one* of the following,
 - Dial-up
 - ❑ RTT = 87 μ s; Link bandwidth: 56 Kbps
 - Satellite
 - ❑ RTT = 230 ms; Link bandwidth: 45 Mbps

Stop-and-Wait

□ Advantage

- Simple
- Achieve reliable transmission on non-reliable medium

□ Disadvantage

- Performance is *poor*
- Could you give an *intuitive* explanation why the performance is *poor*?

Stop-and-Wait

- ❑ Does not keep the pipe full!
 - Q: How much data are needed to keep the pipe full?
 - Product of Delay \times Link Bandwidth
 - ❑ $(1 \times \text{RTT}) \times 10 \text{ Gbps} = 1 \times 40 \text{ ms} \times 10 \text{ Gbps} = 400 \text{ Mb} = 50 \text{ MB}$
 - ❑ $50 \text{ MB} / 1500 \text{ bytes} = 33333 \text{ frames}$
 - $1500 \text{ bytes} \ll \text{the product} \rightarrow \text{low link utilization}$

Link Type	Bandwidth (Typical)	Distance (Typical)	Round-trip Delay	Delay \times BW
Dial-up	56 Kbps	10 km	$87 \mu\text{s}$	5 bits
Wireless LAN	54 Mbps	50 m	$0.33 \mu\text{s}$	18 bits
Satellite	45 Mbps	35,000 km	230 ms	10 Mb
Cross-country fiber	10 Gbps	4,000 km	40 ms	400 Mb

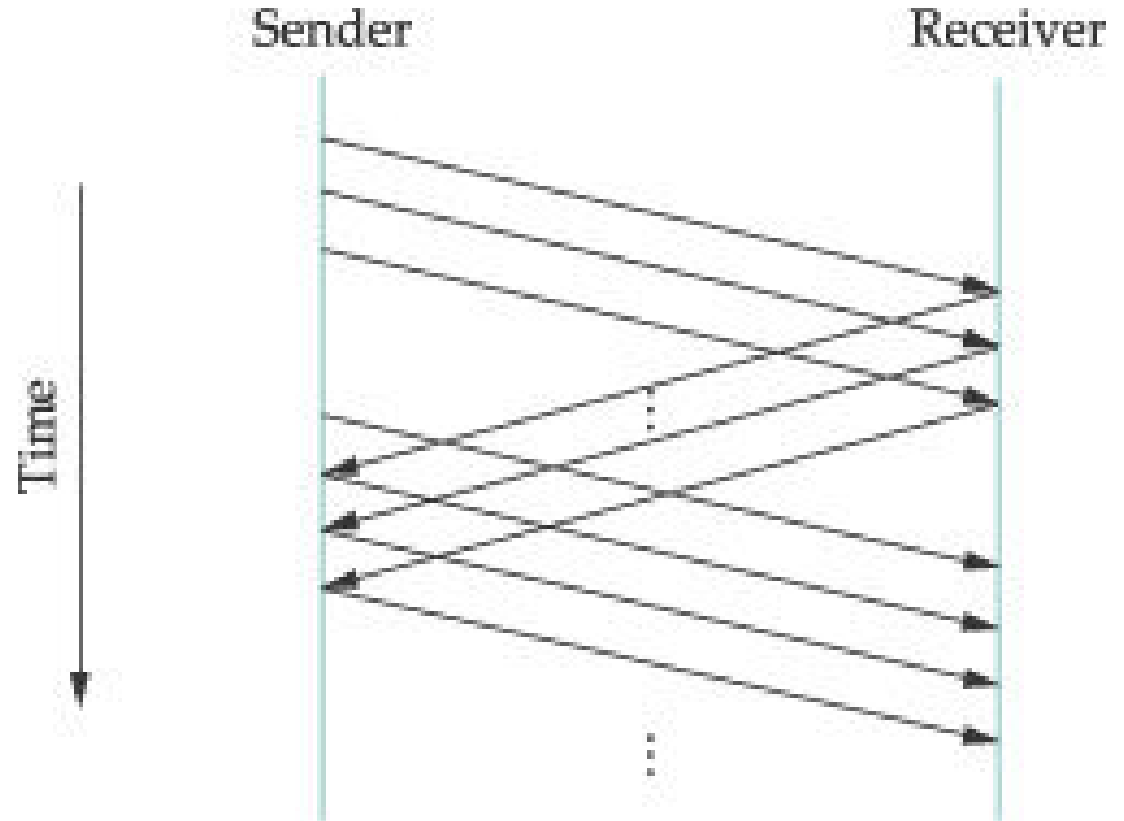
Q: How to keep the pipe full?

How to keep the “pipe” full?

- Free discussion

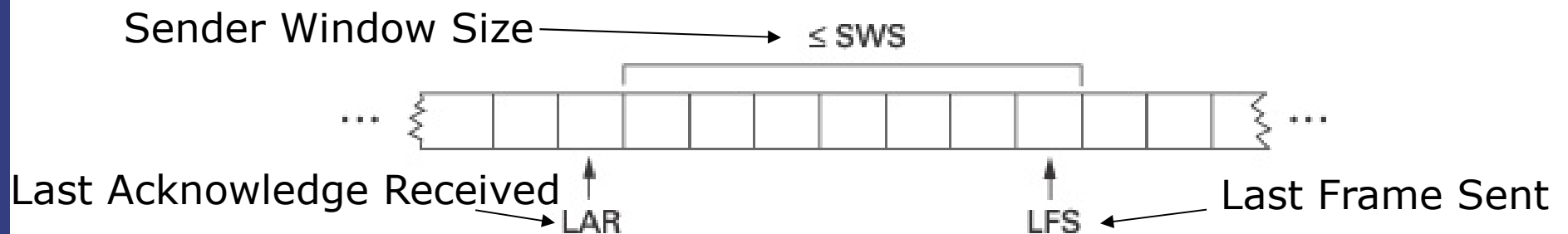
Sliding Window Algorithm

- ❑ Allow multiple unacknowledged frames (send a few frames in a batch) → try to fill the pipe
- ❑ Define a time window (threshold, or upper bound) on unacknowledged frames
 - Sending window
 - Receiving window
- ❑ Have variations



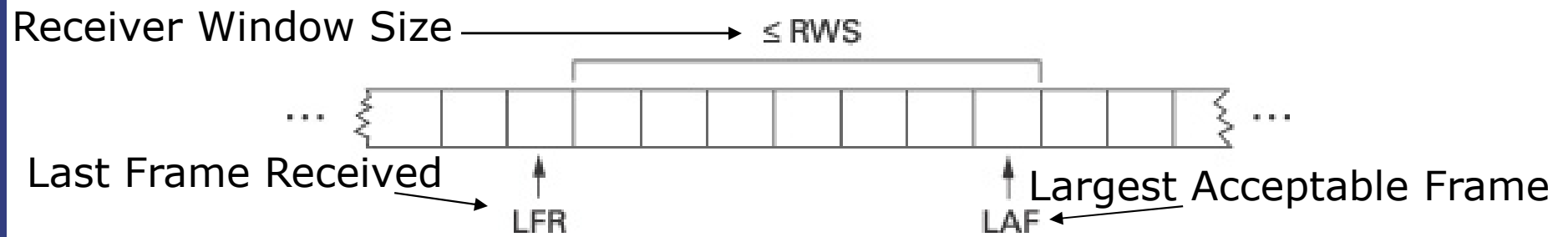
Sliding Windows Algorithm: Sender

- ❑ Assign sequence number to each frame (SeqNum)
- ❑ Maintain three state variables:
 - Send Window Size (SWS)
 - Last Acknowledgment Received (LAR)
 - Last Frame Sent (LFS)
- ❑ Maintain invariant: $LFS - LAR \leq SWS$
- ❑ Advance LAR when ACK arrives
- ❑ Buffer up to SWS frames

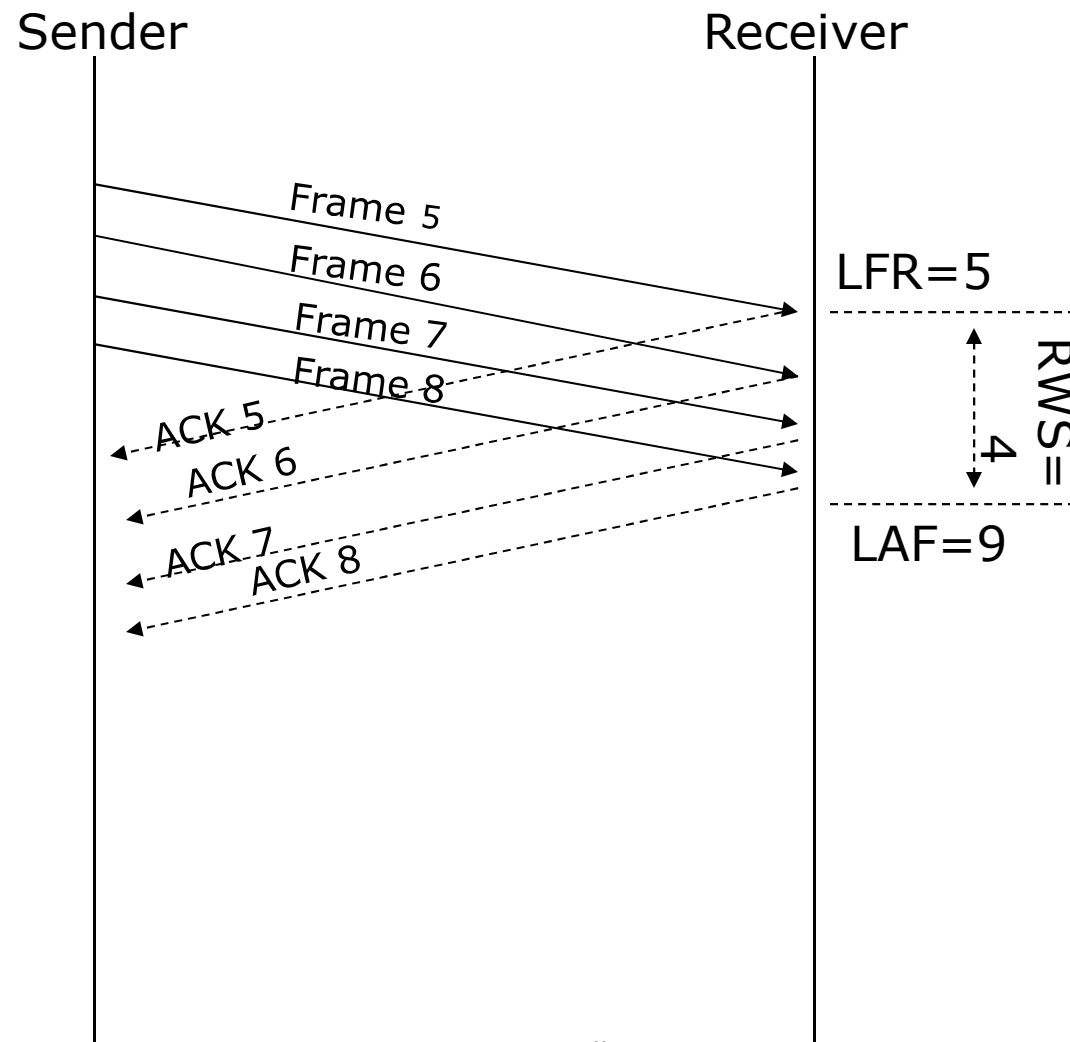


Sliding Windows Algorithm: Receiver

- ❑ Maintain three state variables
 - Receive Window Size (RWS)
 - Largest Acceptable Frame (LAF)
 - Last Frame Received (LFR)
- ❑ Maintain invariant: $LAF - LFR \leq RWS$
- ❑ Frame_{SeqNum} arrives:
 - if $LFR < SeqNum \leq LAF$, accept the frame
 - if $SeqNum \leq LFR$ or $SeqNum > LAF$, discard the frame
- ❑ SeqNumToAck: largest sequence number not yet acknowledged
- ❑ ACK is **cumulative** → ACK all frames with less or equal SeqNum

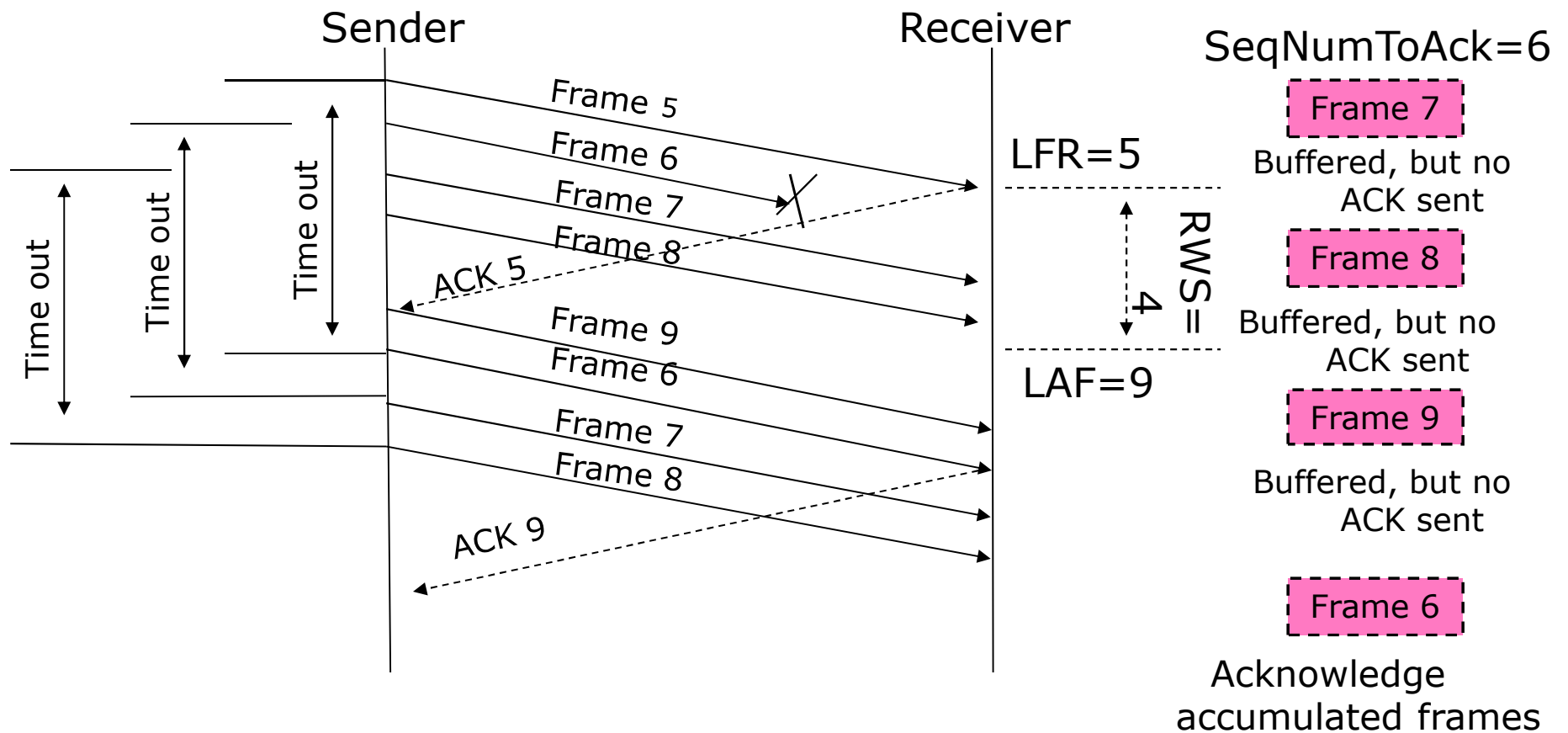


Example: No Frame "Loss"



Example: Frame "Loss"

- Frame 6 is lost



Sliding Window Algorithm: SWS and RWS

- ❑ SWS should be determined by the product of delay \times bandwidth
- ❑ RWS does not have to be equal to SWS
 - RWS = 1, does not buffer any frames that arrive out of order
 - RWS > SWS is meaningless, since it is impossible for more than SWS frames to arrive out of order

Examples

- ❑ Consider following sliding window algorithm
 - Caution: Parameters chosen for demos only. In reality they need to be carefully chosen. Check footnote in page 108.
 - Timeout = $2 \times \text{RTT}$
 - SWS (send window size) = 4
 - ❑ Determined by delay \times bandwidth. Again check footnote in page 108.
 - RWS (receive window size) = 4
- ❑ Show timeline diagrams for the following scenarios
 - Frame 5 lost
 - Frame 6 lost
 - Frames 5-8 lost
 - ACK 6 lost
 - ACK 8 lost and no more frames to send (for an extended period of time)

Sliding Window Algorithm: Implementation – Data Structures

```
typedef u_char  SwpSeqno;
typedef struct {
    SwpSeqno  SeqNum; /* sequence number of this frame */
    SwpSeqno  AckNum; /* ack of received frame */
    u_char    Flags; /* up to 8 bits worth of flags */
} SwpHdr;

typedef struct {
    /* sender side state: */
    SwpSeqno  LAR; /* seqno of last ACK received */
    SwpSeqno  LFS; /* last frame sent */
    Semaphore sendWindowNotFull;
    SwpHdr    hdr; /* pre-initialized header */
    struct sendQ_slot {
        Event  timeout;
                /* event associated with send-timeout */
        Msg    msg;
    } sendQ[SWS];

    /* receiver side state: */
    SwpSeqno  NFE;
                /* seqno of next frame expected */
    struct recvQ_slot {
        int    received; /* is msg valid? */
        Msg    msg;
    } recvQ[RWS];
} SwpState;
```

Sliding Window Algorithm: Implementation – Sending

```
static int
sendSWP(SwpState *state, Msg *frame)
{
    struct sendQ_slot *slot;
    hbuf[HLEN];

    /* wait for send window to open */
    semWait(&state->sendWindowNotFull);
    state->hdr.SeqNum = ++state->LFS;
    slot = &state->sendQ[state->hdr.SeqNum % SWS];
    store_swp_hdr(state->hdr, hbuf);
    msgAddHdr(frame, hbuf, HLEN);
    msgSaveCopy(&slot->msg, frame);
    slot->timeout = evSchedule(swpTimeout, slot,
        SWP_SEND_TIMEOUT);
    return send(LINK, frame);
}
```


Sliding Window Algorithm: Implementation – Receiving (1)

```
static int
deliverSWP(SwpState state, Msg *frame)
{
    SwpHdr   hdr;
    char     *hbuf;

    hbuf = msgStripHdr(frame, HLEN);
    load_swp_hdr(&hdr, hbuf)
    if (hdr->Flags & FLAG_ACK_VALID)
    {
        /* received an acknowledgment---do SENDER side */
        if (swpInWindow(hdr.AckNum, state->LAR + 1,
            state->LFS))
        {
            do
            {
                struct sendQ_slot *slot;
                slot = &state->sendQ[++state->LAR % SWS];
                evCancel(slot->timeout);
                msgDestroy(&slot->msg);
                semSignal(&state->sendWindowNotFull);
            } while (state->LAR != hdr.AckNum);
        }
    }
}
```

Sliding Window Algorithm: Implementation – Receiving (2)

```
if (hdr.Flags & FLAG_HAS_DATA)
{
    struct rcvQ_slot *slot;

    /* received data packet---do RECEIVER side */
    slot = &state->rcvQ[hdr.SeqNum % RWS];
    if (!swpInWindow(hdr.SeqNum, state->NFE,
                    state->NFE + RWS - 1))
    {
        /* drop the message */
        return SUCCESS;
    }
    msgSaveCopy(&slot->msg, frame);
    slot->received = TRUE;

    if (hdr.SeqNum == state->NFE)
    {
        Msg m;

        while (slot->received)
        {
            deliver(HLP, &slot->msg);
            msgDestroy(&slot->msg);
            slot->received = FALSE;
            slot = &state->rcvQ[++state->NFE % RWS];
        }
        /* send ACK: */
        prepare_ack(&m, state->NFE - 1);
        send(LINK, &m);
        msgDestroy(&m);
    }
}
return SUCCESS;
}
```

Exercise L5-2

- Draw a timeline diagram for the sliding window algorithm with $SWS=RWS=3$ frames in the following two situations (draw two time diagrams for each situation). Use a timeout interval of $2 \times RTT$
 - Frame 4 is lost
 - Frame 4-6 are lost

Discussion

- Alternatives or improvement
 - Negative Acknowledgement (NAK)
 - Selective Acknowledgement
- Finite sequence numbers and sliding window
- Frame order and flow control

Summary

- ❑ Reliable delivery
 - Timeout and Acknowledgement
- ❑ Stop-and-Wait
- ❑ Sliding Window
- ❑ Idea: keep the pipe full
 - Many different algorithms exist, e.g., concurrent logical channels
- ❑ How to implement?
 - Consult the book