Chapter 5
Peer-to-Peer Protocols and Data Link Layer

PART I: Peer-to-Peer Protocols
Peer-to-Peer Protocols and Service Models
ARQ Protocols and Reliable Data Transfer
Flow Control
Timing Recovery
TCP Reliable Stream Service & Flow Control
Chapter 5
Peer-to-Peer Protocols and Data Link Layer

PART II: Data Link Controls
Framing
Point-to-Point Protocol
High-Level Data Link Control
Link Sharing Using Statistical Multiplexing
Chapter Overview

- Peer-to-Peer protocols: many protocols involve the interaction between two peers
  - Service Models are discussed & examples given
  - Detailed discussion of ARQ provides example of development of peer-to-peer protocols
  - Flow control, TCP reliable stream, and timing recovery

- Data Link Layer
  - Framing
  - PPP & HDLC protocols
  - Statistical multiplexing for link sharing
Chapter 5
Peer-to-Peer Protocols and Data Link Layer

Peer-to-Peer Protocols and Service Models
Peer-to-Peer Protocols

- **Peer-to-Peer processes** execute layer-n protocol to provide service to layer-(n+1)
- Layer-(n+1) peer calls layer-n and passes Service Data Units (SDUs) for transfer
- Layer-n peers exchange Protocol Data Units (PDUs) to effect transfer
- Layer-n delivers SDUs to destination layer-(n+1) peer
Service Models

- The *service model* specifies the information transfer service layer-\(n\) provides to layer-\((n+1)\)
- The most important distinction is whether the service is:
  - Connection-oriented
  - Connectionless
- Service model possible features:
  - Arbitrary message size or structure
  - Sequencing and Reliability
  - Timing, Pacing, and Flow control
  - Multiplexing
  - Privacy, integrity, and authentication
Connection-Oriented Transfer Service

- Connection Establishment
  - Connection must be established between layer-(n+1) peers
  - Layer-n protocol must: Set initial parameters, e.g. sequence numbers; and Allocate resources, e.g. buffers
- Message transfer phase
  - Exchange of SDUs
- Disconnect phase
- Example: TCP, PPP

Diagram:
- n + 1 peer process send
- Layer n connection-oriented service
- SDU
- n + 1 peer process receive
- SDU
Connectionless Transfer Service

- No Connection setup, simply send SDU
- Each message send independently
- Must provide all address information per message
- Simple & quick
- Example: UDP, IP
Message Size and Structure

- What message size and structure will a service model accept?
  - Different services impose restrictions on size & structure of data it will transfer
  - Single bit? Block of bytes? Byte stream?
  - Ex: Transfer of voice mail = 1 long message
  - Ex: Transfer of voice call = byte stream

(a) 1 voice mail = 1 message = entire sequence of speech samples

(b) 1 call = sequence of 1-byte messages
Segmentation & Blocking

- To accommodate arbitrary message size, a layer may have to deal with messages that are too long or too short for its protocol.

- **Segmentation & Reassembly:** a layer breaks long messages into smaller blocks andreassembles these at the destination.

- **Blocking & Unblocking:** a layer combines small messages into bigger blocks prior to transfer.

![Diagram]

1 long message

2 or more blocks

2 or more short messages

1 block
Reliability & Sequencing

- **Reliability**: Are messages or information stream delivered error-free and without loss or duplication?

- **Sequencing**: Are messages or information stream delivered in order?

- **ARQ protocols** combine error detection, retransmission, and sequence numbering to provide reliability & sequencing

- Examples: TCP and HDLC
Pacing and Flow Control

- Messages can be lost if receiving system does not have sufficient buffering to store arriving messages.
- If destination layer-(n+1) does not retrieve its information fast enough, destination layer-n buffers may overflow.
- *Pacing & Flow Control* provide backpressure mechanisms that control transfer according to availability of buffers at the destination.
- Examples: TCP and HDLC.
Timing

- Applications involving voice and video generate units of information that are related temporally.
- Destination application must reconstruct temporal relation in voice/video units.
- Network transfer introduces delay & jitter.
- Timing Recovery protocols use *timestamps* & *sequence numbering* to control the delay & jitter in delivered information.
- Examples: RTP & associated protocols in Voice over IP.
Multiplexing

- *Multiplexing* enables multiple layer-(n+1) users to share a layer-n service
- A multiplexing tag is required to identify specific users at the destination
- Examples: UDP, IP
Privacy, Integrity, & Authentication

- **Privacy**: ensuring that information transferred cannot be read by others
- **Integrity**: ensuring that information is not altered during transfer
- **Authentication**: verifying that sender and/or receiver are who they claim to be
- **Security protocols** provide these services and are discussed in Chapter 11
- Examples: IPSec, SSL
End-to-End vs. Hop-by-Hop

- A service feature can be provided by implementing a protocol
  - end-to-end across the network
  - across every hop in the network

- Example:
  - Perform error control at every hop in the network or only between the source and destination?
  - Perform flow control between every hop in the network or only between source & destination?

- We next consider the tradeoffs between the two approaches
Error control in Data Link Layer

- Data Link operates over wire-like, directly-connected systems
- Frames can be corrupted or lost, but arrive in order
- Data link performs error-checking & retransmission
- Ensures error-free packet transfer between two systems
Error Control in Transport Layer

- Transport layer protocol (e.g. TCP) sends segments across network and performs end-to-end error checking & retransmission
- Underlying network is assumed to be unreliable
- Segments can experience long delays, can be lost, or arrive out-of-order because packets can follow different paths across network
- End-to-end error control protocol more difficult
End-to-End Approach Preferred

Hop-by-hop cannot ensure E2E correctness

Faster recovery

Simple inside the network

More scalable if complexity at the edge

End-to-end

More scalable if complexity at the edge
Chapter 5
Peer-to-Peer Protocols and Data Link Layer

ARQ Protocols and Reliable Data Transfer
Purpose: to ensure a sequence of information packets is delivered in order and without errors or duplications despite transmission errors & losses

We will look at:

- Stop-and-Wait ARQ
- Go-Back N ARQ
- Selective Repeat ARQ

Basic elements of ARQ:

- Error-detecting code with high error coverage
- ACKs (positive acknowledgments)
- NAKs (negative acknowledgments)
- Timeout mechanism
Stop-and-Wait ARQ

Transmit a frame, wait for ACK
In cases (a) & (b) the transmitting station A acts the same way.
But in case (b) the receiving station B accepts frame 1 twice.
Question: How is the receiver to know the second frame is also frame 1?

Answer: Add frame sequence number in header.

$S_{last}$ is sequence number of most recent transmitted frame.
The transmitting station A misinterprets duplicate ACKs
Incorrectly assumes second ACK acknowledges Frame 1
Question: How is the receiver to know second ACK is for frame 0?
Answer: Add frame sequence number in ACK header
$R_{\text{next}}$ is sequence number of next frame expected by the receiver
Implicitly acknowledges receipt of all prior frames
1-Bit Sequence Numbering Suffices

Global State: \((S_{\text{last}}, R_{\text{next}})\)

Error-free frame 0 arrives at receiver

ACK for frame 1 arrives at transmitter

Error-free frame 1 arrives at receiver

ACK for frame 0 arrives at transmitter
Stop-and-Wait ARQ

Transmitter

**Ready state**
- Await request from higher layer for packet transfer
- When request arrives, transmit frame with updated $S_{last}$ and CRC
- Go to Wait State

**Wait state**
- Wait for ACK or timer to expire; block requests from higher layer
- If timeout expires
  - retract frame and reset timer
- If ACK received:
  - If sequence number is incorrect or if errors detected: ignore ACK
  - If sequence number is correct ($R_{next} = S_{last} + 1$): accept frame, go to Ready state

Receiver

**Always in Ready State**
- Wait for arrival of new frame
- When frame arrives, check for errors
- If no errors detected and sequence number is correct ($S_{last} = R_{next}$), then
  - accept frame,
  - update $R_{next}$
  - send ACK frame with $R_{next}$
  - deliver packet to higher layer
- If no errors detected and wrong sequence number
  - discard frame
  - send ACK frame with $R_{next}$
- If errors detected
  - discard frame
Applications of Stop-and-Wait ARQ

- IBM *Binary Synchronous Communications protocol* (Bisync): character-oriented data link control
- *Xmodem*: modem file transfer protocol
- *Trivial File Transfer Protocol* (RFC 1350): simple protocol for file transfer over UDP
Stop-and-Wait Efficiency

- 10000 bit frame @ 1 Mbps takes 10 ms to transmit
- If wait for ACK = 1 ms, then efficiency = 10/11 = 91%
- If wait for ACK = 20 ms, then efficiency = 10/30 = 33%

Diagram:
- First frame bit enters channel
- Channel idle while transmitter waits for ACK
- Last frame bit enters channel
- Receiver processes frame and prepares ACK
- ACK arrives
- First frame bit arrives at receiver
- Last frame bit arrives at receiver
Stop-and-Wait Model

$t_0 = \text{total time to transmit 1 frame}$

$t_0 = 2t_{\text{prop}} + 2t_{\text{proc}} + t_f + t_{\text{ack}}$

bits/info frame

$= 2t_{\text{prop}} + 2t_{\text{proc}} + \frac{n_f}{R} + \frac{n_a}{R}$

bits/ACK frame

channel transmission rate
Effective transmission rate:

\[ R_{\text{eff}}^0 = \frac{\text{number of information bits delivered to destination}}{\text{total time required to deliver the information bits}} = \frac{n_f - n_o}{t_0}, \]

Transmission efficiency:

\[ \eta_0 = \frac{R_{\text{eff}}}{R} = \frac{n_f - n_o}{t_0 \cdot R} = 1 + \frac{n_a}{n_f} + \frac{2(t_{\text{prop}} + t_{\text{proc}})R}{n_f}, \]

Effect of frame overhead

Effect of ACK frame

Effect of Delay-Bandwidth Product
Example: Impact of Delay-Bandwidth Product

\[ n_f = 1250 \text{ bytes} = 10000 \text{ bits}, \quad n_a = n_o = 25 \text{ bytes} = 200 \text{ bits} \]

<table>
<thead>
<tr>
<th>2xDelayxBW Efficiency</th>
<th>1 ms 200 km</th>
<th>10 ms 2000 km</th>
<th>100 ms 20000 km</th>
<th>1 sec 200000 km</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Mbps</td>
<td>$10^3$ 88%</td>
<td>$10^4$ 49%</td>
<td>$10^5$ 9%</td>
<td>$10^6$ 1%</td>
</tr>
<tr>
<td>1 Gbps</td>
<td>$10^6$ 1%</td>
<td>$10^7$ 0.1%</td>
<td>$10^8$ 0.01%</td>
<td>$10^9$ 0.001%</td>
</tr>
</tbody>
</table>

Stop-and-Wait does not work well for very high speeds or long propagation delays.
S&W Efficiency in Channel with Errors

- Let $1 - P_f = \text{probability frame arrives w/o errors}$
- Avg. # of transmissions to first correct arrival is then $1/ (1 - P_f)$
- "If 1-in-10 get through without error, then avg. 10 tries to success"
- Avg. Total Time per frame is then $t_0/(1 - P_f)$

$$\eta_{SW} = \frac{R_{eff}}{R} = \frac{n_f - n_o}{t_0} \frac{1 - P_f}{1 - P_f} = \frac{1 - n_o}{n_f} \frac{1}{1 + \frac{n_a}{n_f} + 2(t_{prop} + t_{proc})R} (1 - P_f)$$

Effect of frame loss
Example: Impact Bit Error Rate

$n_f=1250$ bytes = 10000 bits, $n_a=n_o=25$ bytes = 200 bits
Find efficiency for random bit errors with $p=0, 10^{-6}, 10^{-5}, 10^{-4}$

$$1 - P_f = (1 - p)^{n_f} \approx e^{-n_f p} $$

for large $n_f$ and small $p$

<table>
<thead>
<tr>
<th>$1 - P_f$ Efficiency</th>
<th>0</th>
<th>$10^{-6}$</th>
<th>$10^{-5}$</th>
<th>$10^{-4}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Mbps &amp; 1 ms</td>
<td>1</td>
<td>0.99</td>
<td>0.905</td>
<td>0.368</td>
</tr>
<tr>
<td></td>
<td>88%</td>
<td>86.6%</td>
<td>79.2%</td>
<td>32.2%</td>
</tr>
</tbody>
</table>

Bit errors impact performance as $n_f p$ approach 1
Go-Back-N

- Improve Stop-and-Wait by not waiting!
- Keep channel busy by continuing to send frames
- Allow a window of up to $W_s$ outstanding frames
- Use $m$-bit sequence numbering
- If ACK for oldest frame arrives before window is exhausted, we can continue transmitting
- If window is exhausted, pull back and retransmit all outstanding frames
- Alternative: Use timeout
Frame transmission are pipelined to keep the channel busy.
Frame with errors and subsequent out-of-sequence frames are ignored.
Transmitter is forced to go back when window of 4 is exhausted.

Go-Back-4:
4 frames are outstanding; so go back 4
Window size long enough to cover round trip time

Stop-and-Wait ARQ

Time-out expires

Receiver is looking for $R_{\text{next}} = 0$

Go-Back-N ARQ

Four frames are outstanding; so go back 4

Receiver is looking for $R_{\text{next}} = 0$
Go-Back-N with Timeout

- Problem with Go-Back-N as presented:
  - If frame is lost and source does not have frame to send, then window will not be exhausted and recovery will not commence

- Use a timeout with each frame
  - When timeout expires, resend all outstanding frames
Go-Back-N Transmitter & Receiver

Receiver

Receiver will only accept a frame that is error-free and that has sequence number $R_{\text{next}}$

When such frame arrives $R_{\text{next}}$ is incremented by one, so the receive window slides forward by one

Transmitter

Frames transmitted and ACKed

Send Window

Buffers

oldest un-ACKed frame

most recent transmission

max Seq # allowed

Timer $S_{\text{last}}$

Timer $S_{\text{last}}+1$

Timer $S_{\text{recent}}$

Timer $S_{\text{last}}+W_s-1$

Receive Window

Frames received

$R_{\text{next}}$
Sliding Window Operation

Transmitter waits for error-free ACK frame with sequence number $S_{\text{last}}$

When such ACK frame arrives, $S_{\text{last}}$ is incremented by one, and the send window slides forward by one.
Maximum Allowable Window Size is $W_s = 2^m - 1$

M = $2^2 = 4$, Go-Back - 4:
Transmitter goes back 4

Receiver has $R_{next} = 0$, but it does not know whether its ACK for frame 0 was received, so it does not know whether this is the old frame 0 or a new frame 0.

M = $2^2 = 4$, Go-Back-3:
Transmitter goes back 3

Receiver has $R_{next} = 3$, so it rejects the old frame 0.
ACK Piggybacking in Bidirectional GBN

Note: Out-of-sequence error-free frames discarded after $R_{next}$ examined
Applications of Go-Back-N ARQ

- *HDLC* (High-Level Data Link Control): bit-oriented data link control
- *V.42 modem*: error control over telephone modem links
Required Timeout & Window Size

- Timeout value should allow for:
  - Two propagation times + 1 processing time: \( 2 \ T_{prop} + T_{proc} \)
  - A frame that begins transmission right before our frame arrives \( T_f \)
  - Next frame carries the ACK, \( T_f \)

- \( W_s \) should be large enough to keep channel busy for \( T_{out} \)
# Required Window Size for Delay-Bandwidth Product

Frame = 1250 bytes = 10,000 bits, \( R = 1 \text{ Mbps} \)

<table>
<thead>
<tr>
<th>( 2(t_{\text{prop}} + t_{\text{proc}}) )</th>
<th>( 2 \times \text{Delay} \times \text{BW} )</th>
<th>Window</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ms</td>
<td>1000 bits</td>
<td>1</td>
</tr>
<tr>
<td>10 ms</td>
<td>10,000 bits</td>
<td>2</td>
</tr>
<tr>
<td>100 ms</td>
<td>100,000 bits</td>
<td>11</td>
</tr>
<tr>
<td>1 second</td>
<td>1,000,000 bits</td>
<td>101</td>
</tr>
</tbody>
</table>
Efficiency of Go-Back-N

- GBN is completely efficient, if $W_s$ large enough to keep channel busy, and if channel is error-free
- Assume $P_f$ frame loss probability, then time to deliver a frame is:
  - $t_f$ if first frame transmission succeeds $(1 - P_f)$
  - $T_f + W_s t_f / (1 - P_f)$ if the first transmission does not succeed

\[
t_{GBN} = t_f (1 - P_f) + P_f \left( t_f + \frac{W_s t_f}{1 - P_f} \right) = t_f + P_f \frac{W_s t_f}{1 - P_f}
\]

\[
\eta_{GBN} = \frac{t_{GBN}}{R} = \frac{n_f - n_o}{1 - n_o} \quad 1 - \frac{n_o}{n_f} (1 - P_f)
\]

Delay-bandwidth product determines $W_s$
Example: Impact Bit Error Rate on GBN

\( n_f = 1250 \, \text{bytes} = 10000 \, \text{bits}, \quad n_a = n_o = 25 \, \text{bytes} = 200 \, \text{bits} \)

Compare S\&W with GBN efficiency for random bit errors with \( p = 0, 10^{-6}, 10^{-5}, 10^{-4} \) and \( R = 1 \, \text{Mbps} & 100 \, \text{ms} \)

\( 1 \, \text{Mbps} \times 100 \, \text{ms} = 100000 \, \text{bits} = 10 \, \text{frames} \rightarrow \text{Use } W_s = 11 \)

<table>
<thead>
<tr>
<th>Efficiency</th>
<th>0</th>
<th>10^{-6}</th>
<th>10^{-5}</th>
<th>10^{-4}</th>
</tr>
</thead>
<tbody>
<tr>
<td>S&amp;W</td>
<td>8.9%</td>
<td>8.8%</td>
<td>8.0%</td>
<td>3.3%</td>
</tr>
<tr>
<td>GBN</td>
<td>98%</td>
<td>88.2%</td>
<td>45.4%</td>
<td>4.9%</td>
</tr>
</tbody>
</table>

- **Go-Back-N** significant improvement over Stop-and-Wait for large delay-bandwidth product
- **Go-Back-N** becomes inefficient as error rate increases
Selective Repeat ARQ

- Go-Back-N ARQ inefficient because *multiple* frames are resent when errors or losses occur.
- Selective Repeat retransmits *only an individual frame*:
  - Timeout causes individual corresponding frame to be resent.
  - NAK causes retransmission of oldest un-acked frame.
- Receiver maintains a *receive window* of sequence numbers that can be accepted:
  - Error-free, but out-of-sequence frames with sequence numbers within the receive window are buffered.
  - Arrival of frame with $R_{\text{next}}$ causes window to slide forward by 1 or more.
Selective Repeat ARQ
Selective Repeat ARQ

Transmitter

Send Window

Frames transmitted and ACKed

$S_{last}$ $S_{recent}$ $S_{last} + W_s - 1$

Buffers

$S_{last}$

$S_{last} + 1$

$S_{recent}$

Receiver

Receive Window

Frames received

$R_{next}$ $R_{next} + W_r - 1$

Buffers

$R_{next} + 1$

$R_{next} + 2$

$R_{next} + W_r - 1$

max Seq # accepted
Send & Receive Windows

Transmitter

Moves $k$ forward when ACK arrives with $R_{next} = S_{last} + k$

$\begin{align*}
    k &= 1, \ldots, W_s - 1
\end{align*}$

Receiver

Moves forward by 1 or more when frame arrives with Seq. # = $R_{next}$
What size $W_s$ and $W_r$ allowed?

- Example: $M=2^2=4$, $W_s=3$, $W_r=3$

![Diagram showing send and receive windows with frame resent and old frame accepted as a new frame in the receive window.](image-url)
**Ws + Wr = 2^m** is maximum allowed

- Example: \( M=2^2=4 \), \( W_s=2 \), \( W_r=2 \)

Old frame 0 rejected because it falls outside the receive window
Why $W_s + W_r = 2^m$ works

- Transmitter sends frames 0 to $W_s-1$; send window empty
- All arrive at receiver
- All ACKs lost
- Transmitter resends frame 0

- Receiver window starts at $\{0, \ldots, W_r\}$
- Window slides forward to $\{W_s, \ldots, W_s + W_r - 1\}$
- Receiver rejects frame 0 because it is outside receive window
Applications of Selective Repeat ARQ

- **TCP** (Transmission Control Protocol): transport layer protocol uses variation of selective repeat to provide reliable stream service

- **Service Specific Connection Oriented Protocol**: error control for signaling messages in ATM networks
Efficiency of Selective Repeat

- Assume $P_f$ frame loss probability, then number of transmissions required to deliver a frame is:
  - $t_f/(1-P_f)$

\[
\eta_{SR} = \frac{n_f - n_o}{t_f/(1-P_f)} = \frac{R}{n_f} = (1 - \frac{n_o}{n_f})(1 - P_f)
\]
Example: Impact Bit Error Rate on Selective Repeat

\[ n_f = 1250 \text{ bytes} = 10000 \text{ bits}, \quad n_a = n_o = 25 \text{ bytes} = 200 \text{ bits} \]

Compare S&W, GBN & SR efficiency for random bit errors with \( p = 0, 10^{-6}, 10^{-5}, 10^{-4} \) and \( R = 1 \text{ Mbps} & 100 \text{ ms} \)

<table>
<thead>
<tr>
<th>Efficiency</th>
<th>0</th>
<th>10^{-6}</th>
<th>10^{-5}</th>
<th>10^{-4}</th>
</tr>
</thead>
<tbody>
<tr>
<td>S&amp;W</td>
<td>8.9%</td>
<td>8.8%</td>
<td>8.0%</td>
<td>3.3%</td>
</tr>
<tr>
<td>GBN</td>
<td>98%</td>
<td>88.2%</td>
<td>45.4%</td>
<td>4.9%</td>
</tr>
<tr>
<td>SR</td>
<td>98%</td>
<td>97%</td>
<td>89%</td>
<td>36%</td>
</tr>
</tbody>
</table>

- Selective Repeat outperforms GBN and S&W, but efficiency drops as error rate increases
Comparison of ARQ Efficiencies

Assume \( n_a \) and \( n_o \) are negligible relative to \( n_f \), and
\[
L = 2(t_{\text{prop}} + t_{\text{proc}})R/n_f = (W_s - 1),
\]
then

Selective-Repeat:
\[
\eta_{SR} = (1 - P_f)(1 - \frac{n_o}{n_f}) \approx (1 - P_f)
\]

Go-Back-N: For \( P_f \approx 0 \), SR & GBN same
\[
\eta_{GBN} = \frac{1 - P_f}{1 + (W_s - 1)P_f} = \frac{1 - P_f}{1 + LP_f}
\]

Stop-and-Wait: For \( P_f \to 1 \), GBN & SW same
\[
\eta_{SW} = \frac{(1 - P_f)}{1 + \frac{n_a}{n_f} + \frac{2(t_{\text{prop}} + t_{\text{proc}})R}{n_f}} \approx \frac{1 - P_f}{1 + L}
\]
ARQ Efficiencies

Delay-Bandwidth product = 10, 100
Chapter 5
Peer-to-Peer Protocols and Data Link Layer

Flow Control
Flow Control

- Receiver has limited buffering to store arriving frames
- Several situations cause buffer overflow
  - Mismatch between sending rate & rate at which user can retrieve data
  - Surges in frame arrivals
- *Flow control* prevents buffer overflow by regulating rate at which source is allowed to send information
Threshold must activate OFF signal while $2 T_{\text{prop}} R$ bits still remain in buffer.
Window Flow Control

- Sliding Window ARQ method with $W_s$ equal to buffer available
  - Transmitter can never send more than $W_s$ frames
- ACKs that slide window forward can be viewed as permits to transmit more
- Can also pace ACKs as shown above
  - Return permits (ACKs) at end of cycle regulates transmission rate
- Problems using sliding window for both error & flow control
  - Choice of window size
  - Interplay between transmission rate & retransmissions
  - TCP separates error & flow control
Chapter 5
Peer-to-Peer Protocols and Data Link Layer

Timing Recovery
Timing Recovery for Synchronous Services

Synchronous source sends periodic information blocks

Network multiplexing & switching introduces random delays
- Packets experience variable transfer delay
- Jitter (variation in interpacket arrival times) also introduced

Applications that involve voice, audio, or video can generate a synchronous information stream

Information carried by equally-spaced fixed-length packets

Networking output not periodic

Timing recovery re-establishes the synchronous nature of the stream
Introduce Playout Buffer

- Delay first packet by maximum network delay
- All other packets arrive with less delay
- Playout packet uniformly thereafter

Packet Arrivals → Playout Buffer → Packet Playout

Sequence numbers help order packets

$T_{max}$
Playout clock must be synchronized to transmitter clock

Receiver too fast; buffer starvation

Receiver too slow; buffer fills and overflows

Many late packets

Playout clock must be synchronized to transmitter clock
Clock Recovery

- Counter attempts to replicate transmitter clock
- Frequency of counter is adjusted according to arriving timestamps
- Jitter introduced by network causes fluctuations in buffer & in local clock
Synchronization to a Common Clock

Clock recovery simple if a common clock is available to transmitter & receiver
- E.g. SONET network clock; Global Positioning System (GPS)
- Transmitter sends $\Delta f$ of its frequency & network frequency
- Receiver adjusts network frequency by $\Delta f$
- Packet delay jitter can be removed completely
Example: Real-Time Protocol

- RTP (RFC 1889) designed to support real-time applications such as voice, audio, video
- RTP provides means to carry:
  - Type of information source
  - Sequence numbers
  - Timestamps
- Actual timing recovery must be done by higher layer protocol
  - MPEG2 for video, MP3 for audio
Chapter 5
Peer-to-Peer Protocols and Data Link Layer

TCP Reliable Stream Service & Flow Control
TCP Reliable Stream Service

Application Layer writes bytes into send buffer through socket

TCP transfers byte stream in order, without errors or duplications

Application Layer reads bytes from receive buffer through socket

Application layer

Transport layer

Transmitter

Send buffer

Segments

Receiver

Receive buffer

Write 45 bytes
Write 15 bytes
Write 20 bytes

Read 40 bytes
Read 40 bytes

ACKs
TCP ARQ Method

- TCP uses *Selective Repeat ARQ*
  - Transfers byte stream without preserving boundaries
- Operates over best effort service of IP
  - Packets can arrive with errors or be lost
  - Packets can arrive out-of-order
  - Packets can arrive after very long delays
  - Duplicate segments must be detected & discarded
  - Must protect against segments from previous connections
- **Sequence Numbers**
  - Seq. # is number of first byte in segment payload
  - Very long Seq. #s (32 bits) to deal with long delays
  - Initial sequence numbers negotiated during connection setup (to deal with very old duplicates)
  - Accept segments within a receive window
Transmitter

Send Window

S_{last} + W_a - 1

S_{last} highest unacknowledged byte
S_{recent} highest-numbered transmitted byte
S_{last} + W_a - 1 highest-numbered byte that can be transmitted
S_{last} + W_s - 1 highest-numbered byte that can be accepted from the application

Receiver

Receive Window

R_{last} + W_R - 1

R_{last} highest-numbered byte not yet read by the application
R_{next} next expected byte
R_{new} highest numbered byte received correctly
R_{last} + W_R - 1 highest-numbered byte that can be accommodated in receive buffer
TCP Connections

- TCP Connection
  - One connection each way
  - Identified uniquely by Send IP Address, Send TCP Port #, Receive IP Address, Receive TCP Port #

- Connection Setup with Three-Way Handshake
  - Three-way exchange to negotiate initial Seq. #'s for connections in each direction

- Data Transfer
  - Exchange segments carrying data

- Graceful Close
  - Close each direction separately
Three Phases of TCP Connection

**Host A**

**Three-way Handshake**
- SYN, Seq_no = x
- SYN, Seq_no = y, ACK, Ack_no = x+1
- Seq_no = x+1, ACK, Ack_no = y+1

**Graceful Close**
- FIN, Seq_no = w
- ACK, Ack_no = w+1

**Data Transfer**

**Host B**
1st Handshake: Client-Server Connection Request

SYN bit set indicates request to establish connection from client to server

Initial Seq. # from client to server
2nd Handshake: ACK from Server

ACK bit set acknowledges connection request; Client-to-Server connection established

ACK Seq. # = Init. Seq. # + 1
### 2nd Handshake: Server-Client Connection Request

**TCP Telnet Capture - Ethereal**

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source IP</th>
<th>Destination IP</th>
<th>Protocol</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.000000</td>
<td>65.95.113.77</td>
<td>128.113.26.22</td>
<td>TCP</td>
<td>2743 &gt; telnet [SYN] Seq=1839733355 Ack=0 Win=31968 Len=0</td>
</tr>
<tr>
<td>2</td>
<td>0.144934</td>
<td>128.113.26.22</td>
<td>65.95.113.77</td>
<td>TCP</td>
<td>2743 &gt; telnet [SYN, ACK] Seq=1877388864 Ack=1839733356 Win=0</td>
</tr>
<tr>
<td>3</td>
<td>0.144270</td>
<td>128.113.26.22</td>
<td>65.95.113.77</td>
<td>TCP</td>
<td>2743 &gt; telnet [ACK] Seq=1839733356 Ack=1877388865 Win=219</td>
</tr>
<tr>
<td>4</td>
<td>0.322432</td>
<td>128.113.26.22</td>
<td>65.95.113.77</td>
<td>TELNET</td>
<td>Telnet Data...</td>
</tr>
<tr>
<td>5</td>
<td>0.323617</td>
<td>65.95.113.77</td>
<td>128.113.26.22</td>
<td>TELNET</td>
<td>Telnet Data...</td>
</tr>
<tr>
<td>6</td>
<td>21.605250</td>
<td>65.95.113.77</td>
<td>128.113.26.22</td>
<td>TCP</td>
<td>2743 &gt; telnet [FIN, ACK] Seq=1839733327 Ack=1877389120 Win=0</td>
</tr>
</tbody>
</table>

**TCP Telnet Capture - Ethereal withHighlighted Details**

- **Initial Seq. # from server to client:**
  - Source IP: 65.95.113.77 (server)
  - Destination IP: 128.113.26.22 (client)
  - Sequence number: 1877388864

- **SYN bit set indicates request to establish connection from server to client:**
  - Flags: 0x0012 (SYN, ACK)
  - Sequence number: 1877388864
  - Acknowledgment number: 1839733356
  - Header length: 20 bytes
  - Window size: 49152

**Internet Protocol**
- Dest Addr: 65.95.113.77 (65.95.113.77)

**Transmission Control Protocol**
- Source Port: telnet (23)
- Destination Port: 2743 (2743)

**TCP Telnet Capture - Ethereal with Highlighted Details**

- Flags: 0x0012 (SYN, ACK)
  - 1. = Syn: Set

**TCP Telnet Capture - Ethereal with Highlighted Details**

- Flags: 0x0012 (SYN, ACK)
  - 1. = Syn: Set

**TCP Telnet Capture - Ethereal with Highlighted Details**

- Flags: 0x0012 (SYN, ACK)
  - 1. = Syn: Set

**TCP Telnet Capture - Ethereal with Highlighted Details**

- Flags: 0x0012 (SYN, ACK)
  - 1. = Syn: Set
3rd Handshake: ACK from Client

ACK Seq. # = Init. Seq. # + 1

ACK bit set acknowledges connection request; Connections in both directions established
TCP Data Exchange

- Application Layers write bytes into buffers
- TCP sender forms segments
  - When bytes exceed threshold or timer expires
  - Upon PUSH command from applications
  - Consecutive bytes from buffer inserted in payload
  - Sequence # & ACK # inserted in header
  - Checksum calculated and included in header
- TCP receiver
  - Performs selective repeat ARQ functions
  - Writes error-free, in-sequence bytes to receive buffer
Data Transfer: Server-to-Client Segment

12 bytes of payload
Push set
12 bytes of payload carries telnet option negotiation
Graceful Close: Client-to-Server Connection

Client initiates closing of its connection to server.
Graceful Close: Client-to-Server Connection

Server ACKs request; client-to-server connection closed

ACK Seq. # = Previous Seq. # + 1
Flow Control

- TCP receiver controls rate at which sender transmits to prevent buffer overflow.
- TCP receiver advertises a window size specifying number of bytes that can be accommodated by receiver,
  \[ W_A = W_R - (R_{\text{new}} - R_{\text{last}}) \]
- TCP sender obliged to keep # outstanding bytes below \( W_A \)
  \[ (S_{\text{recent}} - S_{\text{last}}) \leq W_A \]
TCP window flow control

<table>
<thead>
<tr>
<th>Time</th>
<th>Sequence Number</th>
<th>Acknowledgment Number</th>
<th>Window Size</th>
<th>Data Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>1</td>
<td>2000</td>
<td>2048</td>
<td>None</td>
</tr>
<tr>
<td>$t_1$</td>
<td>2000</td>
<td>1</td>
<td>1024</td>
<td>2000-3023</td>
</tr>
<tr>
<td>$t_2$</td>
<td>3024</td>
<td>1</td>
<td>1024</td>
<td>3024-4047</td>
</tr>
<tr>
<td>$t_3$</td>
<td>129</td>
<td>4048</td>
<td>1024</td>
<td>4048-4559</td>
</tr>
<tr>
<td>$t_4$</td>
<td>4048</td>
<td>129</td>
<td>1024</td>
<td>4048-4559</td>
</tr>
</tbody>
</table>
TCP Retransmission Timeout

- TCP retransmits a segment after timeout period
  - Timeout too short: excessive number of retransmissions
  - Timeout too long: recovery too slow
  - Timeout depends on RTT: time from when segment is sent to when ACK is received
- Round trip time (RTT) in Internet is highly variable
  - Routes vary and can change in mid-connection
  - Traffic fluctuates
- TCP uses adaptive estimation of RTT
  - Measure RTT each time ACK received: $\tau_n$

\[
\tau_{RTT}(\text{new}) = \alpha \tau_{RTT}(\text{old}) + (1 - \alpha) \tau_n
\]

- $\alpha = 7/8$ typical
RTT Variability

- Estimate variance $\sigma^2$ of RTT variation
- Estimate for timeout:
  \[ t_{out} = t_{RTT} + k \sigma_{RTT} \]
- If RTT highly variable, timeout increase accordingly
- If RTT nearly constant, timeout close to RTT estimate

- Approximate estimation of deviation
  \[ d_{RTT}(new) = \beta d_{RTT}(old) + (1-\beta) | \tau_n - t_{RTT} | \]

\[ t_{out} = t_{RTT} + 4 d_{RTT} \]
Chapter 5
Peer-to-Peer Protocols and Data Link Layer

PART II: Data Link Controls
Framing
Point-to-Point Protocol
High-Level Data Link Control
Link Sharing Using Statistical Multiplexing
Data Link Protocols

- Directly connected, wire-like
- Losses & errors, but no out-of-sequence frames
- Applications: Direct Links; LANs; Connections across WANs

Data Links Services
- Framing
- Error control
- Flow control
- Multiplexing
- Link Maintenance
- Security: Authentication & Encryption

Examples
- PPP
- HDLC
- Ethernet LAN
- IEEE 802.11 (Wi Fi) LAN
Chapter 5
Peer-to-Peer Protocols and Data Link Layer

Framing
Framing

- Mapping stream of physical layer bits into frames
- Mapping frames into bit stream
- Frame boundaries can be determined using:
  - Character Counts
  - Control Characters
  - Flags
  - CRC Checks
Character-Oriented Framing

Data to be sent

A DLE B ETX DLE STX E

After stuffing and framing

DLE STX A DLE DLE B ETX DLE DLE STX E DLE ETX

- Frames consist of integer number of bytes
  - Asynchronous transmission systems using ASCII to transmit printable characters
  - Octets with HEX value <20 are nonprintable
- Special 8-bit patterns used as control characters
  - STX (start of text) = 0x02; ETX (end of text) = 0x03;
- Byte used to carry non-printable characters in frame
  - DLE (data link escape) = 0x10
  - DLE STX (DLE ETX) used to indicate beginning (end) of frame
  - Insert extra DLE in front of occurrence of DLE STX (DLE ETX) in frame
  - All DLEs occur in pairs except at frame boundaries
Framing & Bit Stuffing

HDLC frame

<table>
<thead>
<tr>
<th>Flag</th>
<th>Address</th>
<th>Control</th>
<th>Information</th>
<th>FCS</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>any number of bits</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Frame delineated by flag character
- HDLC uses *bit stuffing* to prevent occurrence of flag 01111110 inside the frame
- Transmitter inserts extra 0 after each consecutive five 1s *inside* the frame
- Receiver checks for five consecutive 1s
  - if next bit = 0, it is removed
  - if next two bits are 10, then flag is detected
  - If next two bits are 11, then frame has errors
Example: Bit stuffing & de-stuffing

(a) Data to be sent

0110111111111100

After stuffing and framing

011111110011011111011111000001111110

(b) Data received

01111111000011101111101111101110011111110

After destuffing and deframing

*000111011111-11111-110*
PPP Frame

- PPP uses similar frame structure as HDLC, except
  - Protocol type field
  - Payload contains an integer number of bytes
- PPP uses the same flag, but uses byte stuffing
- Problems with PPP byte stuffing
  - Size of frame varies unpredictably due to byte insertion
  - Malicious users can inflate bandwidth by inserting 7D & 7E
Byte-Stuffing in PPP

- PPP is character-oriented version of HDLC
- Flag is 0x7E (01111110)
- Control escape 0x7D (01111101)
- Any occurrence of flag or control escape inside of frame is replaced with 0x7D followed by original octet XORed with 0x20 (00100000)

Data to be sent

| 41 | 7D | 42 | 7E | 50 | 70 | 46 |

After stuffing and framing

| 7E | 41 | 7D | 5D | 42 | 7D | 5E | 50 | 70 | 46 | 7E |
## Generic Framing Procedure

<table>
<thead>
<tr>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>0-60</th>
<th>GFP payload area</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLI</td>
<td>cHEC</td>
<td>Type</td>
<td>tHEC</td>
<td>GEH</td>
<td>GFP payload</td>
</tr>
<tr>
<td>Payload length indicator</td>
<td>Core header error checking</td>
<td>Payload type</td>
<td>Type header error checking</td>
<td>GFP extension headers</td>
<td>GFP payload</td>
</tr>
</tbody>
</table>

- GFP combines frame length indication with CRC
  - PLI indicated length of frame, then simply count characters
  - cHEC (CRC-16) protects against errors in count field (single-bit error correction + error detection)
- GFP designed to operate over octet-synchronous physical layers (e.g. SONET)
  - Frame-mapped mode for variable-length payloads: Ethernet
  - Transparent mode carries fixed-length payload: storage devices
GFP Synchronization & Scrambling

- Synchronization in three-states
  - **Hunt state**: examine 4-bytes to see if CRC ok
    - If no, move forward by one-byte
    - If yes, move to pre-sync state
  - **Pre-sync state**: tentative PLI indicates next frame
    - If N successful frame detections, move to sync state
    - If no match, go to hunt state
  - **Sync state**: normal state
    - Validate PLI/cHEC, extract payload, go to next frame
    - Use single-error correction
    - Go to hunt state if non-correctable error

- Scrambling
  - Payload is scrambled to prevent malicious users from inserting long strings of 0s which cause SONET equipment to lose bit clock synchronization (as discussed in line code section)
Chapter 5
Peer-to-Peer Protocols and Data Link Layer

*Point-to-Point Protocol*
PPP: Point-to-Point Protocol

- Data link protocol for point-to-point lines in Internet
  - Router-router; dial-up to router
1. Provides *Framing and Error Detection*
  - Character-oriented HDLC-like frame structure
2. *Link Control Protocol*
  - Bringing up, testing, bringing down lines; negotiating options
  - **Authentication**: key capability in ISP access
3. A family of *Network Control Protocols* specific to different network layer protocols
  - IP, OSI network layer, IPX (Novell), Appletalk
PPP Applications

PPP used in many point-to-point applications

- Telephone Modem Links 30 kbps
- Packet over SONET 600 Mbps to 10 Gbps
  - IP→PPP→SONET

PPP is also used over shared links such as Ethernet to provide LCP, NCP, and authentication features
  - PPP over Ethernet (RFC 2516)
  - Used over DSL
PPP Frame Format

- PPP can support multiple network protocols simultaneously
- Specifies what kind of packet is contained in the payload
  - e.g. LCP, NCP, IP, OSI CLNP, IPX...

PPP Frame Structure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag</td>
<td>01111110</td>
</tr>
<tr>
<td>Address</td>
<td>1111111</td>
</tr>
<tr>
<td>Control</td>
<td>00000011</td>
</tr>
<tr>
<td>Protocol</td>
<td></td>
</tr>
<tr>
<td>Information</td>
<td></td>
</tr>
<tr>
<td>FCS</td>
<td></td>
</tr>
<tr>
<td>CRC 16 or CRC 32</td>
<td></td>
</tr>
</tbody>
</table>

- HDLC
- Unnumbered frame
- All stations are to accept the frame
PPP Phases

1. Carrier detected
2. Options negotiated
3. Authentication completed
4. NCP configuration
5. Data transport, e.g. send/receive IP packets
6. NCP used to tear down the network layer connection (free up IP address); LCP used to shut down data link layer connection
7. Modem hangs up

Home PC to Internet Service Provider
1. PC calls router via modem
2. PC and router exchange LCP packets to negotiate PPP parameters
3. Check on identities
4. NCP packets exchanged to configure the network layer, e.g. TCP/IP (requires IP address assignment)
5. Data transport, e.g. send/receive IP packets
6. NCP used to tear down the network layer connection (free up IP address); LCP used to shut down data link layer connection
7. Modem hangs up
PPP Authentication

- **Password Authentication Protocol**
  - Initiator must send ID & password
  - Authenticator replies with authentication success/fail
  - After several attempts, LCP closes link
  - Transmitted unencrypted, susceptible to eavesdropping

- **Challenge-Handshake Authentication Protocol (CHAP)**
  - Initiator & authenticator share a secret key
  - Authenticator sends a challenge (random # & ID)
  - Initiator computes cryptographic checksum of random # & ID using the shared secret key
  - Authenticator also calculates cryptographic checksum & compares to response
  - Authenticator can reissue challenge during session
Example: PPP connection setup in dialup modem to ISP
Chapter 5
Peer-to-Peer Protocols and Data Link Layer

High-Level Data Link Control
High-Level Data Link Control (HDLC)

- Bit-oriented data link control
- Derived from IBM Synchronous Data Link Control (SDLC)
- Related to Link Access Procedure Balanced (LAPB)
  - LAPD in ISDN
  - LAPM in cellular telephone signaling
HDLC Data Transfer Modes

- **Normal Response Mode**
  - Used in polling multidrop lines

- **Asynchronous Balanced Mode**
  - Used in full-duplex point-to-point links

- Mode is selected during connection establishment
### HDLC Frame Format

<table>
<thead>
<tr>
<th>Flag</th>
<th>Address</th>
<th>Control</th>
<th>Information</th>
<th>FCS</th>
<th>Flag</th>
</tr>
</thead>
</table>

- Control field gives HDLC its functionality
- Codes in fields have specific meanings and uses
  - Flag: delineate frame boundaries
  - Address: identify *secondary* station (1 or more octets)
    - In ABM mode, a station can act as primary or secondary so address changes accordingly
  - Control: purpose & functions of frame (1 or 2 octets)
  - Information: contains user data; length not standardized, but implementations impose maximum
  - Frame Check Sequence: 16- or 32-bit CRC
### Control Field Format

#### Information Frame

<table>
<thead>
<tr>
<th>1</th>
<th>2-4</th>
<th>5</th>
<th>6-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N(S)</td>
<td>P/F</td>
<td>N(R)</td>
</tr>
</tbody>
</table>

- **S**: Supervisory Function Bits
- **N(R)**: Receive Sequence Number
- **N(S)**: Send Sequence Number
- **P/F**: Poll/final bit used in interaction between primary and secondary

#### Supervisory Frame

| 1 | 0 | S | S | P/F | N(R) |

- **S**: Supervisory Function Bits
- **N(R)**: Receive Sequence Number
- **P/F**: Poll/final bit used in interaction between primary and secondary

#### Unnumbered Frame

| 1 | 1 | M | M | P/F | M | M | M |

- **M**: Unnumbered Function Bits
Information frames

- Each I-frame contains sequence number N(S)
- Positive ACK piggybacked
  - N(R)=Sequence number of next frame expected
    acknowledges all frames up to and including N(R)-1
- 3 or 7 bit sequence numbering
  - Maximum window sizes 7 or 127
- Poll/Final Bit
  - NRM: Primary polls station by setting P=1; Secondary sets F=1 in last I-frame in response
  - Primaries and secondaries always interact via paired P/F bits
Error Detection & Loss Recovery

- Frames lost due to loss-of-synch or receiver buffer overflow
- Frames may undergo errors in transmission
- CRCs detect errors and such frames are treated as lost
- Recovery through ACKs, timeouts & retransmission
- Sequence numbering to identify out-of-sequence & duplicate frames
- HDLC provides for options that implement several ARQ methods
Supervisory frames

Used for error (ACK, NAK) and flow control (Don’t Send):

- **Receive Ready (RR), SS=00**
  - ACKs frames up to N(R)-1 when piggyback not available

- **REJECT (REJ), SS=01**
  - Negative ACK indicating N(R) is first frame not received correctly. Transmitter must resend N(R) and later frames

- **Receive Not Ready (RNR), SS=10**
  - ACKs frame N(R)-1 & requests that no more I-frames be sent

- **Selective REJECT (SREJ), SS=11**
  - Negative ACK for N(R) requesting that N(R) be selectively retransmitted
Unnumbered Frames

- Setting of Modes:
  - SABM: Set Asynchronous Balanced Mode
  - UA: acknowledges acceptance of mode setting commands
  - DISC: terminates logical link connection

- Information Transfer between stations
  - UI: Unnumbered information

- Recovery used when normal error/flow control fails
  - FRMR: frame with correct FCS but impossible semantics
  - RSET: indicates sending station is resetting sequence numbers

- XID: exchange station id and characteristics
Connection Establishment & Release

- Supervisory frames used to establish and release data link connection
- In HDLC
  - Set Asynchronous Balanced Mode (SABM)
  - Disconnect (DISC)
  - Unnumbered Acknowledgment (UA)
Example: HDLC using NRM (polling)

A polls B

B, RR, 0, P

B, SREJ, 1
C, RR, 0, P

B, SREJ, 1, P

B, I, 0, 5

A rejects fr1

A polls C

C, RR, 0, F

A polls B, requests selective retrans. fr1

B, I, 0, 0
B, I, 1, 0
B, I, 2, 0, F
B, I, 3, 0
B, I, 4, 0, F

A send info fr0 to B, ACKs up to 4

B sends 3 info frames

B resends fr1

Then fr 3 & 4

C nothing to send

N(S) N(R)

Address of secondary

Time

N(R)
Frame Exchange using Asynchronous Balanced Mode

<table>
<thead>
<tr>
<th>Combined Station A</th>
<th>Combined Station B</th>
</tr>
</thead>
<tbody>
<tr>
<td>B, I, 0, 0</td>
<td>A, I, 0, 0</td>
</tr>
<tr>
<td>B, I, 1, 0</td>
<td>A, I, 1, 1</td>
</tr>
<tr>
<td>B, I, 2, 1</td>
<td>A, I, 2, 1</td>
</tr>
<tr>
<td>B, I, 3, 2</td>
<td>B, REJ, 1</td>
</tr>
<tr>
<td>B, I, 4, 3</td>
<td>A, I, 3, 1</td>
</tr>
<tr>
<td>B, I, 1, 3</td>
<td>B, RR, 2</td>
</tr>
<tr>
<td>B, I, 2, 4</td>
<td>B, RR, 3</td>
</tr>
<tr>
<td>B, I, 3, 4</td>
<td></td>
</tr>
</tbody>
</table>

- B sends 5 frames
- B goes back to 1

- A ACKs fr0
- A rejects fr1
- A ACKs fr1
- A ACKs fr2
Flow Control

- Flow control is required to prevent transmitter from overrunning receiver buffers
- Receiver can control flow by delaying acknowledgement messages
- Receiver can also use supervisory frames to explicitly control transmitter
  - Receive Not Ready (RNR) & Receive Ready (RR)
Chapter 5
Peer-to-Peer Protocols and Data Link Layer

Link Sharing Using Statistical Multiplexing
Statistical Multiplexing

- Multiplexing concentrates bursty traffic onto a shared line
- Greater efficiency and lower cost
**Tradeoff Delay for Efficiency**

(a) **Dedicated lines**

- Dedicated lines involve not waiting for other users, but lines are used inefficiently when user traffic is bursty

(b) **Shared lines**

- Shared lines concentrate packets into shared line; packets buffered (delayed) when line is not immediately available

- Dedicated lines involve not waiting for other users, but lines are used inefficiently when user traffic is bursty
- Shared lines concentrate packets into shared line; packets buffered (delayed) when line is not immediately available
Multiplexers inherent in Packet Switches

- Packets/frames forwarded to buffer prior to transmission from switch
- Multiplexing occurs in these buffers
Multiplexer Modeling

- Arrivals: What is the packet interarrival pattern?
- Service Time: How long are the packets?
- Service Discipline: What is order of transmission?
- Buffer Discipline: If buffer is full, which packet is dropped?

- Performance Measures:
  - Delay Distribution; Packet Loss Probability; Line Utilization
Delay = Waiting + Service Times

- Packets arrive and wait for service
- Waiting Time: from arrival instant to beginning of service
- Service Time: time to transmit packet
- Delay: total time in system = waiting time + service time
Fluctuations in Packets in the System

(a) Dedicated lines

(b) Shared line

(c) $N(t)$

Number of packets in the system
Packet Lengths & Service Times

- $R$ bits per second transmission rate
- $L = \#$ bits in a packet
- $X = L/R = \text{time to transmit ("service") a packet}$
- Packet lengths are usually variable
  - Distribution of lengths $\rightarrow$ Dist. of service times
  - Common models:
    - Constant packet length (all the same)
    - Exponential distribution
    - Internet Measured Distributions fairly constant
      - See next chart
Measure Internet Packet Distribution

- Dominated by TCP traffic (85%)
- ~40% packets are minimum-sized 40 byte packets for TCP ACKs
- ~15% packets are maximum-sized Ethernet 1500 frames
- ~15% packets are 552 & 576 byte packets for TCP implementations that do not use path MTU discovery
- Mean=413 bytes
- Stand Dev=509 bytes
- Source: caida.org
M/M/1/K Queueing Model

Poisson Arrivals rate $\lambda$

$K - 1$ buffer

Exponential service time with rate $\mu$

At most $K$ customers allowed in system

- $1$ customer served at a time; up to $K - 1$ can wait in queue
- Mean service time $E[X] = 1/\mu$
- Key parameter Load: $\rho = \lambda/\mu$
- When $\lambda \ll \mu$ ($\rho \approx 0$), customers arrive infrequently and usually find system empty, so delay is low and loss is unlikely
- As $\lambda$ approaches $\mu$ ($\rho \rightarrow 1$), customers start bunching up and delays increase and losses occur more frequently
- When $\lambda > \mu$ ($\rho > 0$), customers arrive faster than they can be processed, so most customers find system full and those that do enter have to wait about $K - 1$ service times
Poisson Arrivals

- Average Arrival Rate: $\lambda$ packets per second
- Arrivals are equally-likely to occur at any point in time
- Time between consecutive arrivals is an exponential random variable with mean $1/\lambda$
- Number of arrivals in interval of time $t$ is a Poisson random variable with mean $\lambda t$

$$P[k \text{ arrivals in } t \text{ seconds}] = \frac{(\lambda t)^k}{k!} e^{-\lambda t}$$
Exponential Distribution

\[ P[X > t] = e^{-t/E[X]} = e^{-\lambda t} \quad \text{for } t > 0. \]
Probability of Overflow:

\[ P_{\text{loss}} = \frac{(1 - \rho)\rho^K}{1 - \rho^{K+1}} \]

Average Total Packet Delay:

\[ E[N] = \frac{\rho}{1 - \rho} - \frac{(K + 1)\rho^{K+1}}{1 - \rho^{K+1}} \]

\[ E[T] = \frac{E[N]}{\lambda(1 - P_K)} \]
M/M/1/10

- Maximum 10 packets allowed in system
- Minimum delay is 1 service time
- Maximum delay is 10 service times
- At 70% load delay & loss begin increasing
- What if we add more buffers?
M/M/1 Queue

- $P_b = 0$ since customers are never blocked
- Average Time in system $E[T] = E[W] + E[X]$
- When $\lambda \ll \mu$, customers arrive infrequently and delays are low
- As $\lambda$ approaches $\mu$, customers start bunching up and average delays increase
- When $\lambda > \mu$, customers arrive faster than they can be processed and queue grows without bound (unstable)

Poisson Arrivals rate $\lambda$ → Infinite buffer → Exponential service time with rate $\mu$

*Unlimited number of customers allowed in system*
Avg. Delay in M/M/1 & M/D/1

\[ E[T_M] = \frac{1}{\lambda} \left[ \frac{\rho}{1 - \rho} \right] = \left[ \frac{1}{1 - \rho} \right] \frac{1}{\mu} = \left[ \frac{\rho}{1 - \rho} \right] \frac{1}{\mu} + \frac{1}{\mu} \quad \text{for M/M/1 model.} \]

\[ E[T_D] = \left[ 1 + \frac{\rho}{2(1 - \rho)} \right] \frac{1}{\mu} = \left[ \frac{\rho}{2(1 - \rho)} \right] \frac{1}{\mu} + \frac{1}{\mu} \quad \text{for M/D/1 system.} \]
Effect of Scale

- $C = 100,000$ bps
- Exp. Dist. with Avg. Packet Length: 10,000 bits
- Service Time: $X=0.1$ second
- Arrival Rate: 7.5 pkts/sec
- Load: $\rho=0.75$
- Mean Delay:
  \[ E[T] = \frac{0.1}{1-0.75} = 0.4 \text{ sec} \]

- $C = 10,000,000$ bps
- Exp. Dist. with Avg. Packet Length: 10,000 bits
- Service Time: $X=0.001$ second
- Arrival Rate: 750 pkts/sec
- Load: $\rho=0.75$
- Mean Delay:
  \[ E[T] = \frac{0.001}{1-0.75} = 0.004 \text{ sec} \]
  Reduction by factor of 100

Aggregation of flows can improve Delay & Loss Performance
Example: Header overhead & Goodput

- Let \( R = 64 \) kbps
- Assume IP+TCP header = 40 bytes
- Assume constant packets of total length
  - \( L = 200, 400, 800, 1200 \) bytes
- Find avg. delay vs. goodput (information transmitted excluding header overhead)

- Service rate \( \mu = \frac{64000}{8L} \) packets/second
- Total load \( \rho = \frac{\lambda}{64000} \frac{8}{8L} \)
- Goodput = \( \lambda \) packets/sec x 8(L-40) bits/packet
- Max Goodput = \( (1-40/L)64000 \) bps
Header overhead limits maximum goodput
Burst Multiplexing / Speech Interpolation

- Voice active < 40% time
- **No buffering**, on-the-fly switch bursts to available trunks
- Can handle 2 to 3 times as many calls
- Tradeoff: Trunk Utilization vs. Speech Loss
  - Fractional Speech Loss: fraction of active speech lost
- Demand Characteristics
  - Talkspurt and Silence Duration Statistics
  - Proportion of time speaker active/idle
Speech Loss vs. Trunks

Typical requirement

speech loss = \sum_{k=m+1}^{n} \binom{n}{k} p^k (1 - p)^{n-k} \quad \text{where} \quad \binom{n}{k} = \frac{n!}{k!(n-k)!}.
Effect of Scale

- Larger flows lead to better performance
- Multiplexing Gain = # speakers / # trunks

*Trunks required for 1% speech loss*

<table>
<thead>
<tr>
<th>Speakers</th>
<th>Trunks</th>
<th>Multiplexing Gain</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>13</td>
<td>1.85</td>
<td>0.74</td>
</tr>
<tr>
<td>32</td>
<td>16</td>
<td>2.00</td>
<td>0.80</td>
</tr>
<tr>
<td>40</td>
<td>20</td>
<td>2.00</td>
<td>0.80</td>
</tr>
<tr>
<td>48</td>
<td>23</td>
<td>2.09</td>
<td>0.83</td>
</tr>
</tbody>
</table>
Packet Speech Multiplexing

Many voice terminals generating voice packets

Buffer

Buffer overflow

- Digital speech carried by fixed-length packets
- No packets when speaker silent
- Synchronous packets when speaker active
- Buffer packets & transmit over shared high-speed line
- Tradeoffs: Utilization vs. Delay/Jitter & Loss
Packet Switching of Voice

- Packetization delay: time for speech samples to fill a packet
- Jitter: variable inter-packet arrivals at destination
- Playback strategies required to compensate for jitter/loss
  - Flexible delay inserted to produce fixed end-to-end delay
  - Need buffer overflow/underflow countermeasures
  - Need clock recovery algorithm
Chapter 5
Peer-to-Peer Protocols and Data Link Layer

ARQ Efficiency Calculations
Stop & Wait Performance

1 successful transmission \( i - 1 \) unsuccessful transmissions

\[
E[t_{\text{total}}] = t_0 + \sum_{i=1}^{\infty} (i - 1)t_{\text{out}} P[n_i = i]
\]

\[
= t_0 + \sum_{i=1}^{\infty} (i - 1)t_{\text{out}} (1 - P_f)^{i-1} P_f
\]

\[
= t_0 + \frac{t_{\text{out}} P_f}{1 - P_f} = t_0 \frac{1}{1 - P_f}.
\]

Efficiency:

\[
\eta_{SW} = \frac{n_f - n_o}{E[t_{\text{total}}]} = \frac{1 - n_o}{R} = \frac{1 - n_o}{n_f} \cdot \frac{2(t_{\text{prop}} + t_{\text{proc}})R}{1 + n_a + \frac{2(t_{\text{prop}} + t_{\text{proc}})R}{n_f}}
\]

\[
= (1 - P_f)n_0.
\]
Go-Back-N Performance

1 successful transmission  

\[ E[t_{total}] = t_f + \sum_{i=1}^{\infty} (i-1)W_s t_f P[n_t = i] \]

\[ = t_f + W_s t_f \sum_{i=1}^{\infty} (i-1)(1-P_f)^{i-1} P_f \]

\[ = t_f + \frac{W_s t_f P_f}{1-P_f} = t_f \frac{1+(W_s-1)P_f}{1-P_f}. \]

Efficiency:

\[ \eta_{GBN} = \frac{n_f - n_o}{E[t_{total}]/R} = (1-P_f) \frac{1-n_o}{n_f} \frac{1}{1+(W_s-1)P_f}. \]