

# CIS 525 - Telecommunications and Data Communications Systems

## Lecture 24: Session Layer

Mitch Neilsen  
[neilsen@cis.ksu.edu](mailto:neilsen@cis.ksu.edu)



# Overview

- Transport Layer Protocols
  - User Datagram Protocol (UDP) (5.1)
  - Transmission Control Protocol (TCP) (5.2)
  - Congestion Control (Ch. 6)
    - Queuing Disciplines
    - Responding to Congestion
    - Avoiding Congestion
- Session Layer Protocols
  - Roll-back Recovery
  - Remote Procedure Call (5.3)
- Presentation Layer Protocols (Ch. 7)



# TRANSPORT LAYER

- The **Transport Layer** provides reliable, end-to-end communication (TCP) and congestion control.
- **Congestion Control**
  - Respond to congestion when it occurs
  - Additive Increase/Multiplicative Decrease (AIMD)
- **Congestion Avoidance**
  - Try to predict when congestion will occur, and avoid it

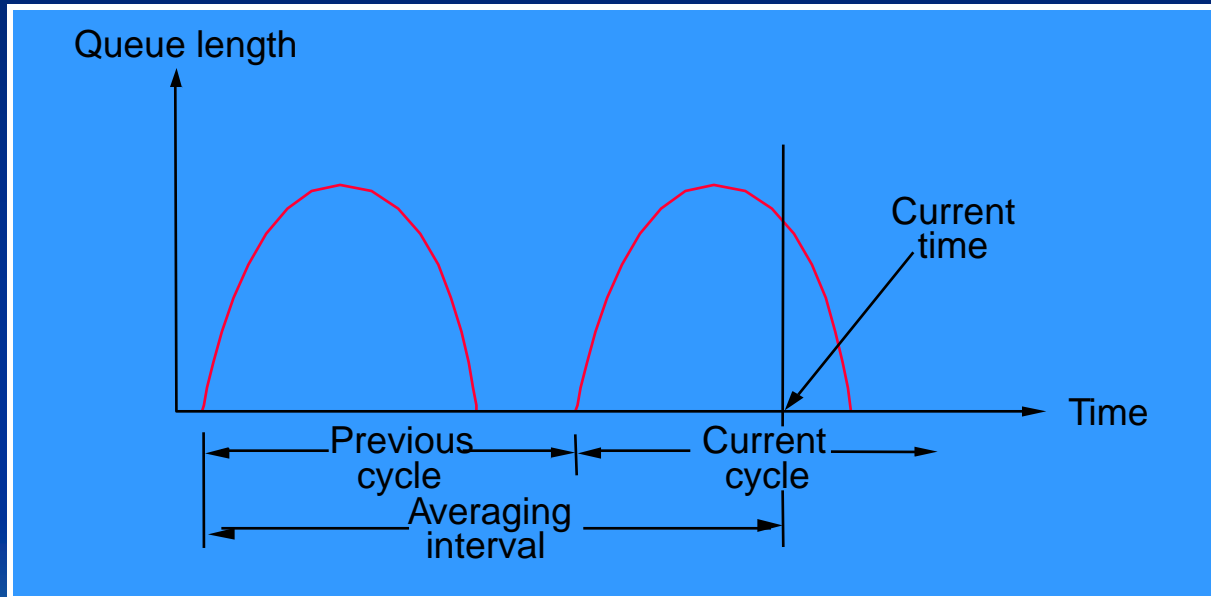


# Congestion Avoidance

- TCP's strategy – Congestion Control
  - control congestion once it happens
  - repeatedly increase load in an effort to find the point at which congestion occurs, and then back off
- Alternative strategy
  - **predict** when congestion is about to happen
  - reduce rate before packets start being discarded
  - This is called congestion **avoidance**, instead of congestion *control*
- **Two possibilities**
  - router-centric: DECbit and RED Gateways
  - host-centric: TCP Vegas

# DECbit

- Add a binary congestion bit to each packet header
- Router
  - monitors average queue length over last busy+idle cycle



- set congestion bit if average queue length  $> 1$
- attempts to balance throughput against delay

# End Hosts

- Destination echoes bit back to source
- Source records how many packets resulted in set bit
- If less than 50% of last window's worth had bit set
  - increase `CongestionWindow` by 1 packet
- If 50% or more of last window's worth had bit set
  - decrease `CongestionWindow` by 0.875 times



# Random Early Detection (RED)

- Notification is implicit
  - just drop the packet (TCP will timeout)
  - could make explicit by marking the packet
- Early random drop
  - rather than wait for queue to become full, drop each arriving packet with some *drop probability* whenever the queue length exceeds some *drop level*



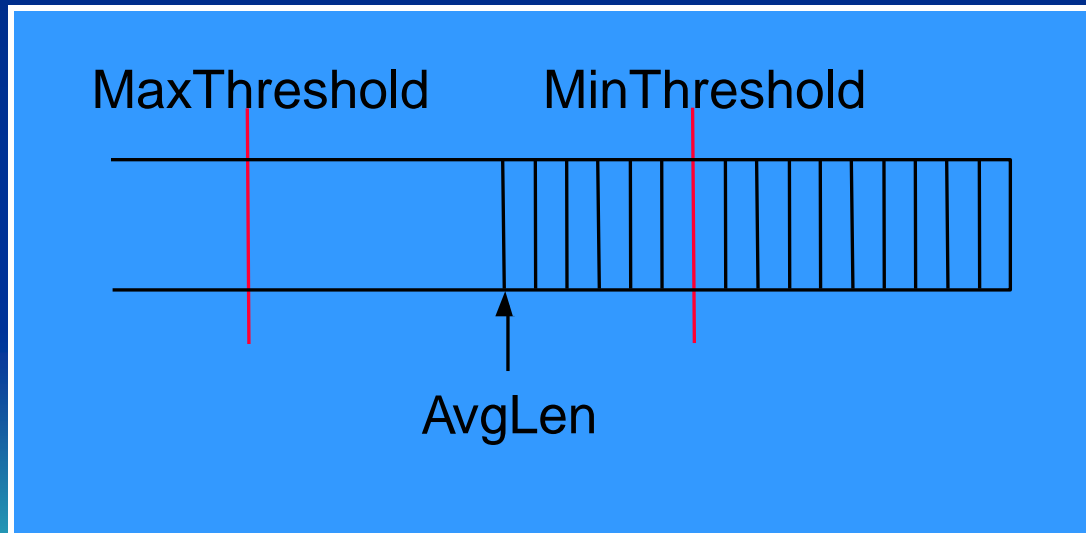
# RED Details

- Compute average queue length

$$\text{AvgLen} = (1 - \text{Weight}) * \text{AvgLen} + \text{Weight} * \text{SampleLen}$$

$0 < \text{Weight} < 1$  (usually 0.002)

`SampleLen` is queue length each time a packet arrives



# RED Details (cont)

- Two queue length thresholds

```
if AvgLen <= MinThreshold then
```

```
    enqueue the packet
```

```
if MinThreshold < AvgLen < MaxThreshold then
```

```
    calculate probability P
```

```
    drop arriving packet with probability P
```

```
if MaxThreshold <= AvgLen then
```

```
    drop arriving packet
```



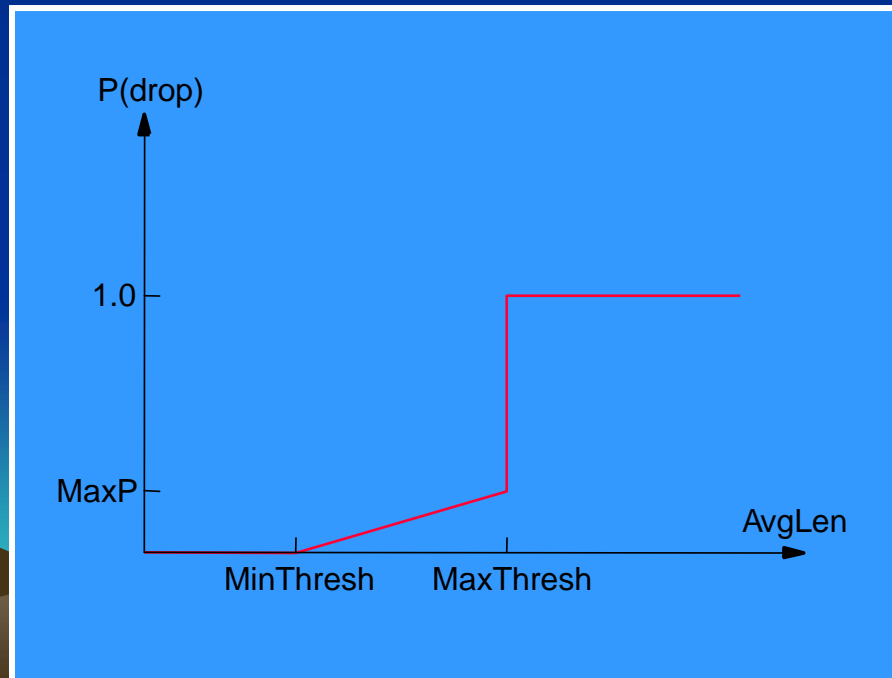
# RED Details (cont)

- Computing probability P

$$\text{TempP} = \text{MaxP} * (\text{AvgLen} - \text{MinThreshold}) / (\text{MaxThreshold} - \text{MinThreshold})$$

$$P = \text{TempP} / (1 - \text{count} * \text{TempP})$$

- Drop Probability Curve

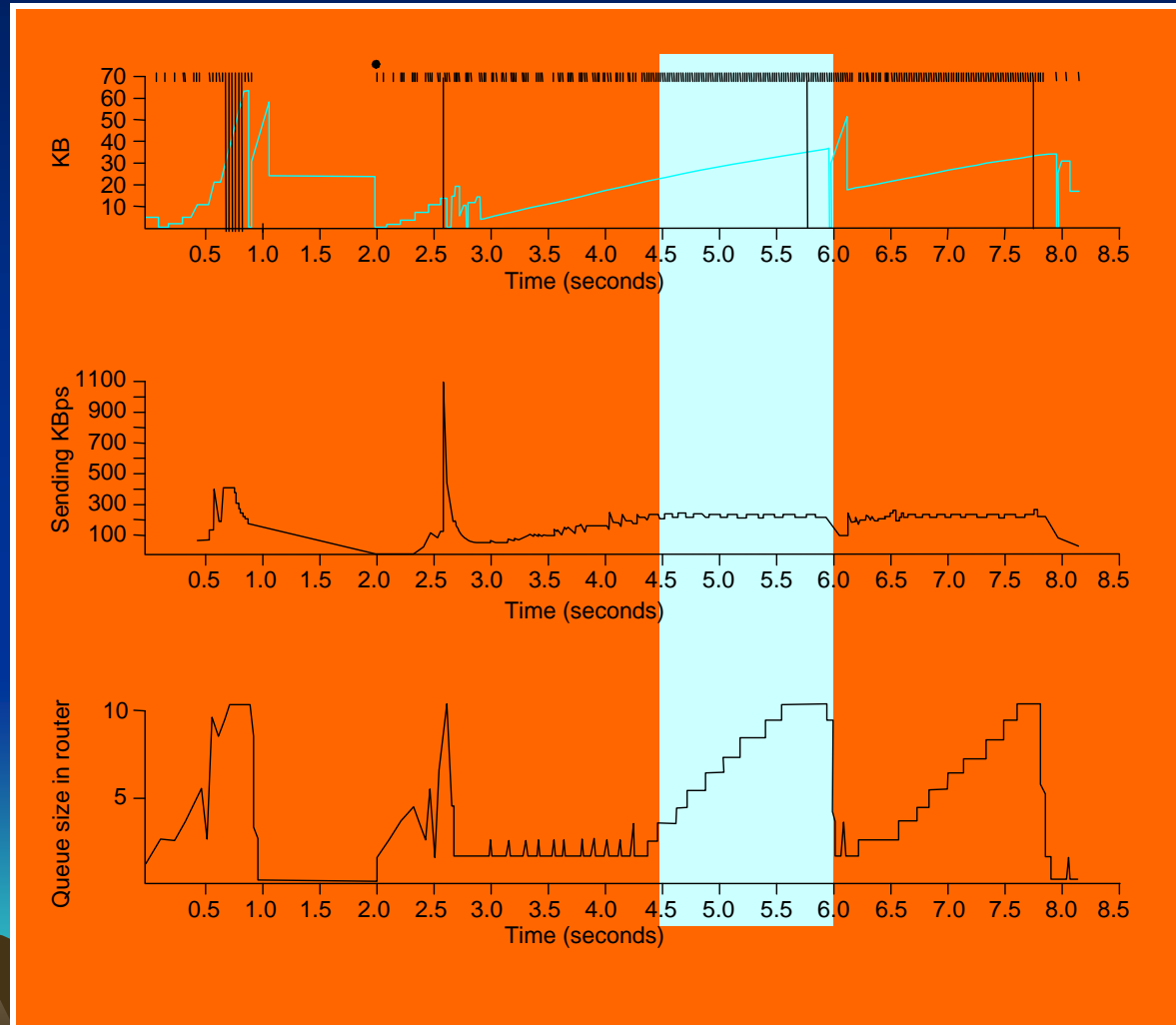


# Tuning RED

- Probability of dropping a particular flow's packet(s) is roughly proportional to the share of the bandwidth that flow is currently getting
- **MaxP** is typically set to 0.02, meaning that when the average queue size is halfway between the two thresholds, the gateway drops roughly one out of 50 packets.
- If traffic is bursty, then **MinThreshold** should be sufficiently large to allow link utilization to be maintained at an acceptably high level
- Difference between two thresholds should be larger than the typical increase in the calculated average queue length in one RTT; setting **MaxThreshold** to twice **MinThreshold** is reasonable for traffic on today's Internet
- Penalty Box for Offenders

# TCP Vegas

- Idea: **source** watches for some sign that router's queue is building up and congestion will occur soon; e.g.,
  - RTT grows
  - sending rate flattens



# Algorithm

- **BaseRTT** is set to be the minimum of all measured RTTs (commonly the RTT of the first packet)
- If the connection is not overflowing, then  
    **ExpectedRate** = **CongestionWindow**/**BaseRTT**
- Source calculates sending rate (**ActualRate**) once per RTT
- Source compares **ActualRate** with **ExpectedRate**

```
Diff = ExpectedRate - ActualRate
```

```
if Diff <  $\alpha$ 
```

```
    increase CongestionWindow linearly
```

```
else if Diff >  $\beta$ 
```

```
    decrease CongestionWindow linearly
```

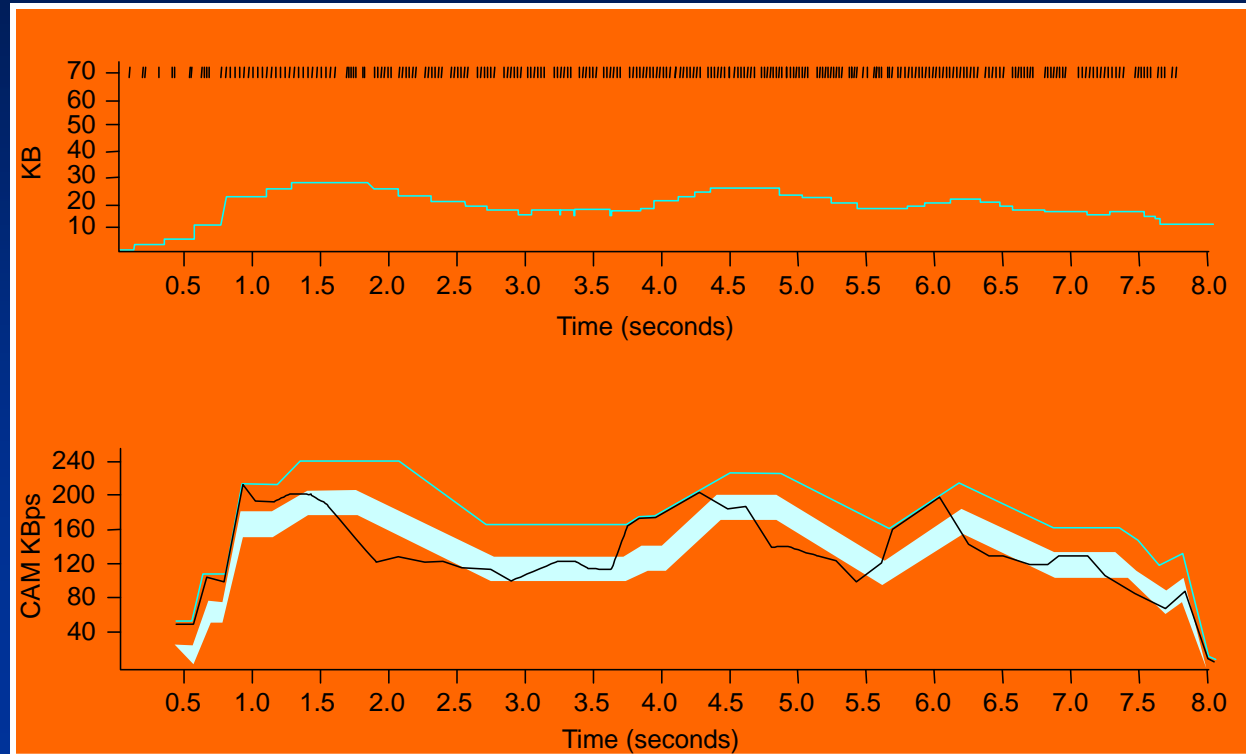
```
else
```

```
    leave CongestionWindow unchanged
```



# Algorithm (cont)

- Parameters
  - $\alpha = 1$  packet
  - $\beta = 3$  packets



- Even faster retransmit
  - keep fine-grained timestamps for each packet
  - check for timeout on first duplicate ACK

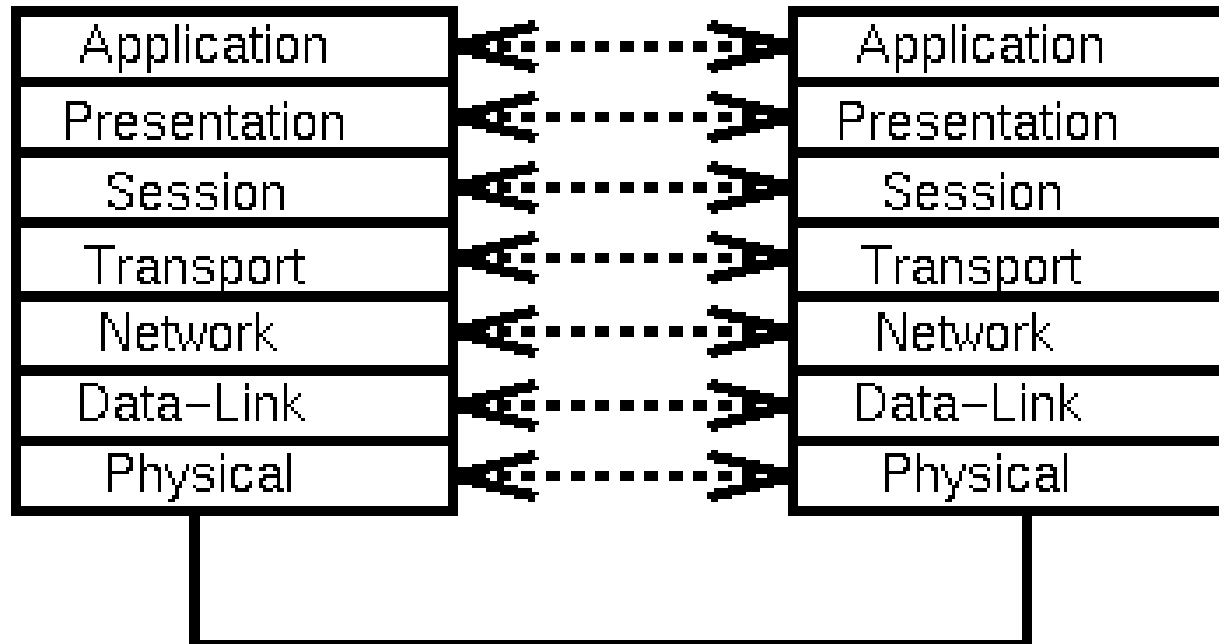
# SESSION LAYER

- The **Session Layer** consists of value-added services built on top of the Transport Layer:
  - Roll-back Recovery
  - Remote Login Sessions
  - Remote Procedure Calls



# OSI 7-Layer Reference Model

OSI 7-Layer Reference Model [1984, ISO]



# Session

- A **session** goes through three phases:
  - establishment
  - use
  - release

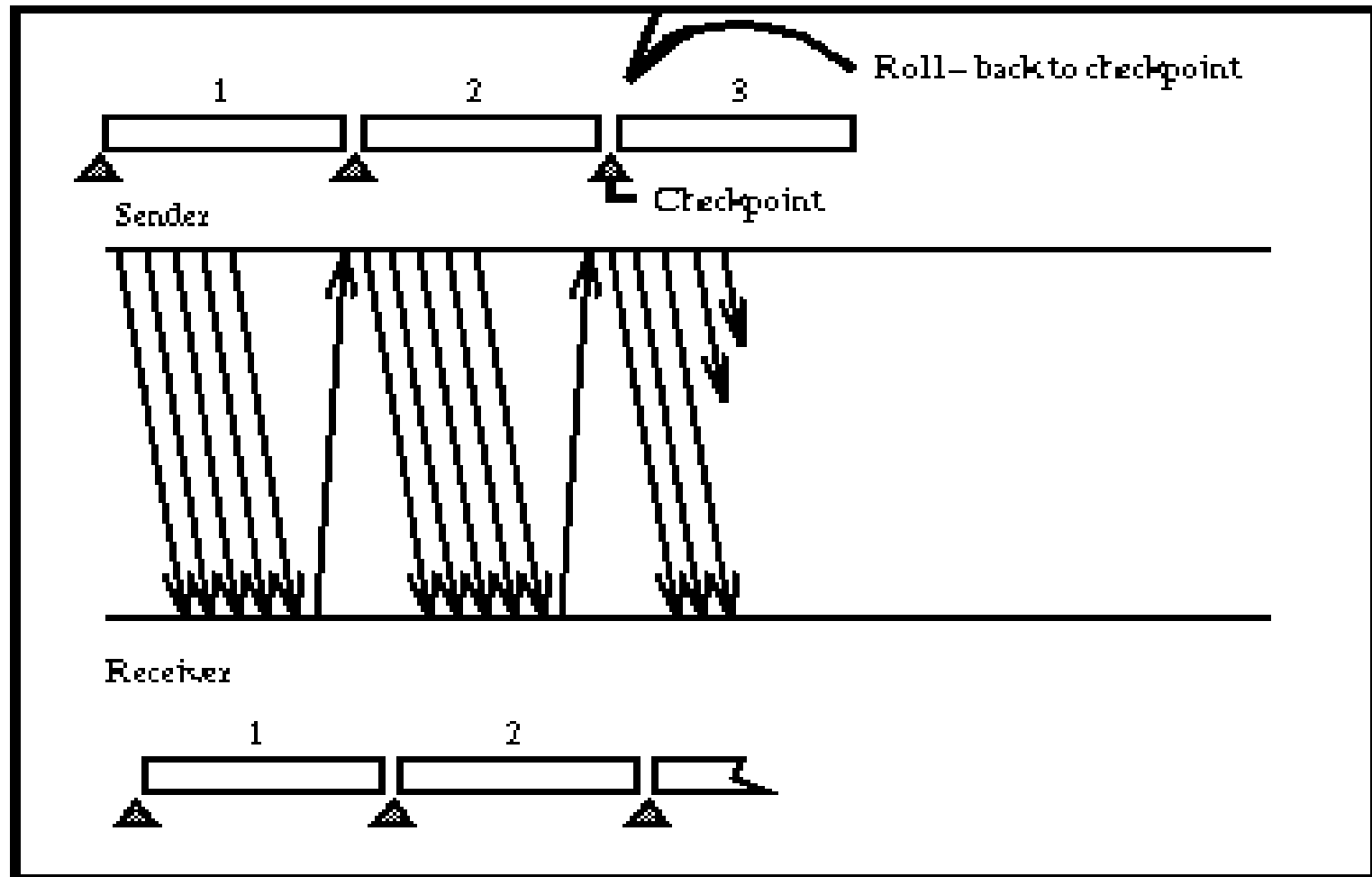


# Roll-Back Recovery

- **Checkpoints** are used to mark known consistent states.
- When a failure occurs, the session can be rolled back to the previous checkpoint.
- Example: Advanced file transfer protocols use roll-back recovery.



# Roll-Back Recovery

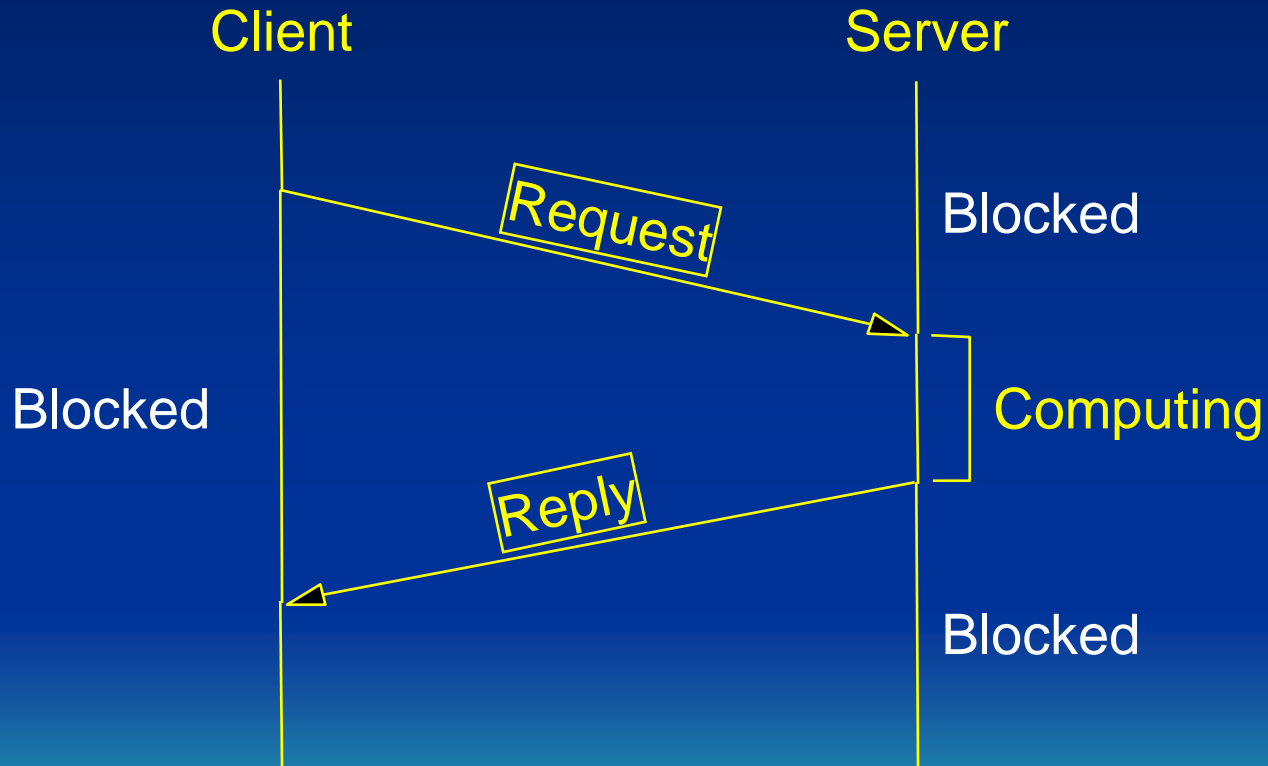


# Remote Procedure Call (RPC)

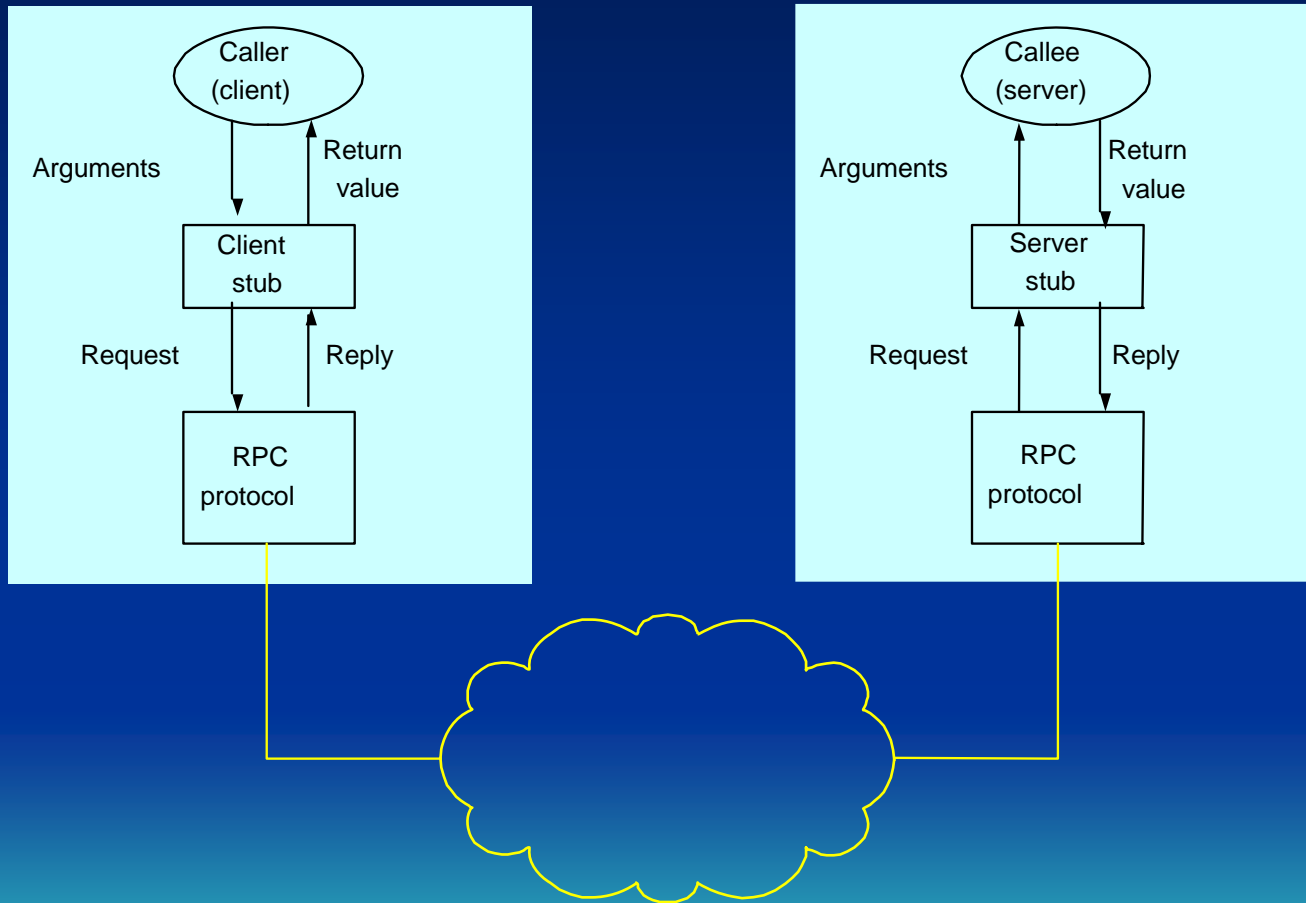
- A **remote procedure call** is used to allow procedures on a remote machine to be called from the local machine.
- The RPC protocol can be classified as a session layer protocol.
- Remote procedure calls are more complex because the local machine is independent from the remote machine and the machines have no shared memory to pass arguments.



# RPC Timeline

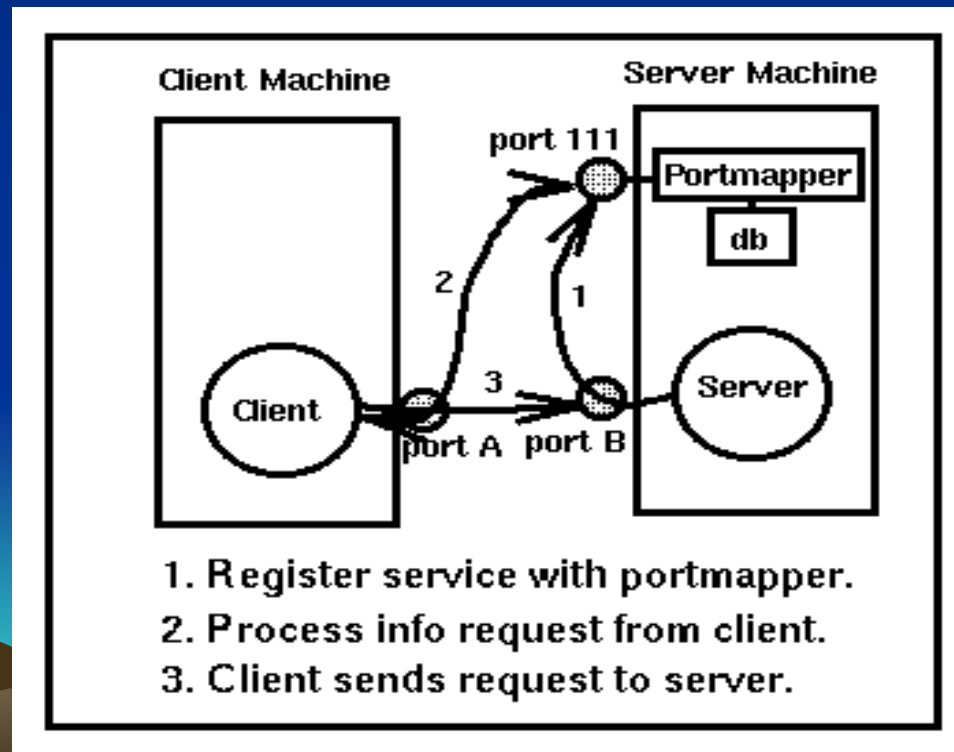


# RPC Components



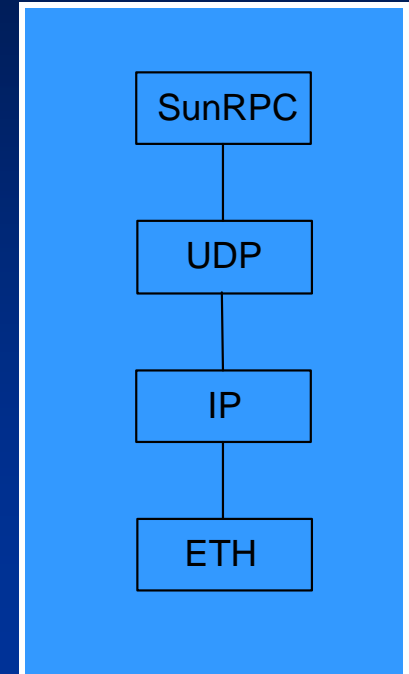
# Establishing an RPC Session

- The server must register its services (procedures) with the **portmapper**. The client must contact the portmapper to determine if the requested service (e.g., program and procedure) is available; and if so, on which port.



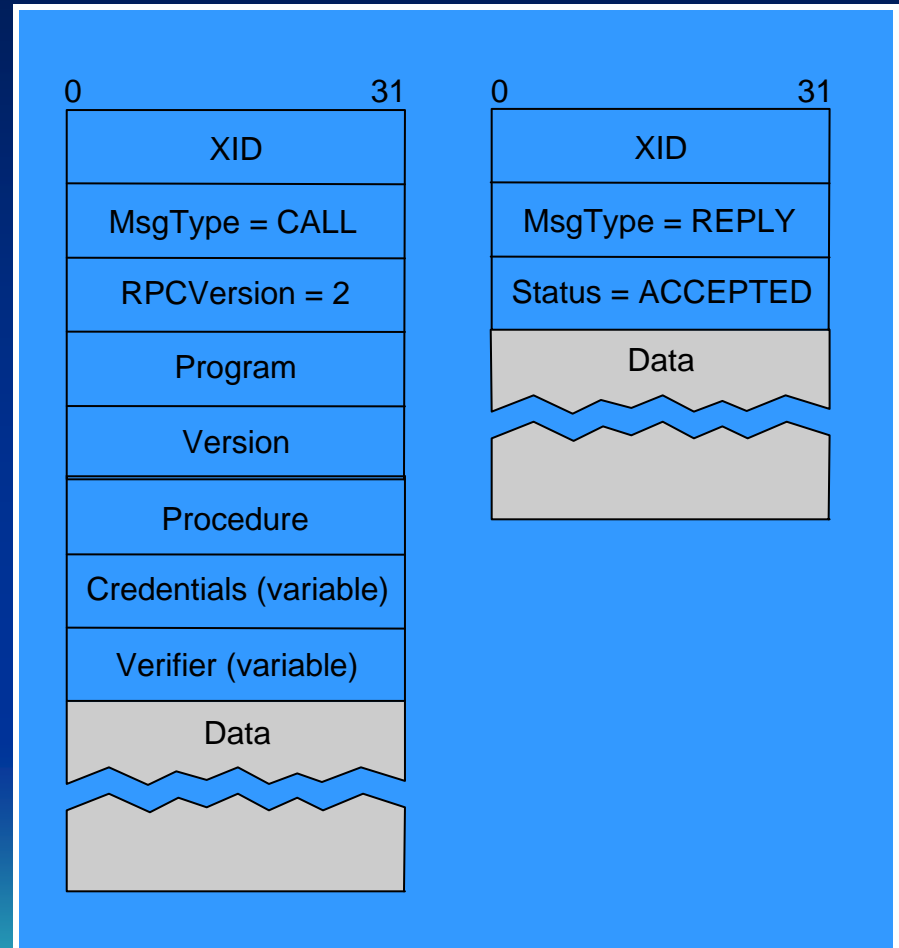
# SunRPC

- IP implements BLAST-equivalent
- SunRPC implements CHAN-equivalent
- UDP + SunRPC implement SELECT-equivalent
  - UDP dispatches to program (ports bound to programs)
  - SunRPC dispatches requests to procedures within the program



# SunRPC Header Format

- **XID** = transaction id is similar to CHAN's MID
- Server does not remember last XID it serviced
- Problem if client retransmits request while reply is in transit



# PRESENTATION LAYER

- The **Presentation Layer** preserves the **meaning** of the information transmitted.
- There are three major issues:
  1. Representation
  2. Compression
  3. Encryption



# 1. Representation

- Data transmitted between hosts must be **represented** in a meaningful format.
- Methods:
  - network canonical form (XDR, ASN.1)
  - receiver-makes-right (NDR)



# 2. Compression

- Data sent over a channel can be viewed as a sequence of symbols:  $S_1, S_2, \dots, S_n$ .
  - The set of symbols is **finite**.
  - The symbols are used with **differing frequencies**.
  - The symbols are used in **different contexts**.
- Data compression is used to transform the symbols into a more compact representation. In this way, less data is transmitted.



# Standard Fixed-Length Encoding

- Assign a sequence number to each element.
- Only use the fact that the set of symbols is finite.
- Example, a library could use the Library of Congress Number instead of the title to represent each book.



# Frequency Dependent Coding

- Assign symbols that occur with a **higher frequency** a shorter code.
- In the English language, some letters occur with a much higher frequency than others. For example, the letter "E" occurs about 100 times more often than "Q"; so, we should use a shorter encoding for "E".



# Letter Frequency Distribution

## Frequency Distribution of Letters

A - 8.0	N - 7.0
B - 1.5	O - 8.0
C - 3.0	P - 2.0
D - 4.0	Q - 0.2
E - 13.0	R - 6.5
F - 2.0	S - 6.0
G - 1.5	T - 9.0
H - 6.0	U - 3.0
I - 6.5	V - 1.0
J - 0.5	W - 1.5
K - 0.5	X - 0.5
L - 3.5	Y - 2.0
M - 3.0	Z - 0.2

# Entropy

- **Entropy** is the theoretical lower bound on the average number of bits required per symbol.

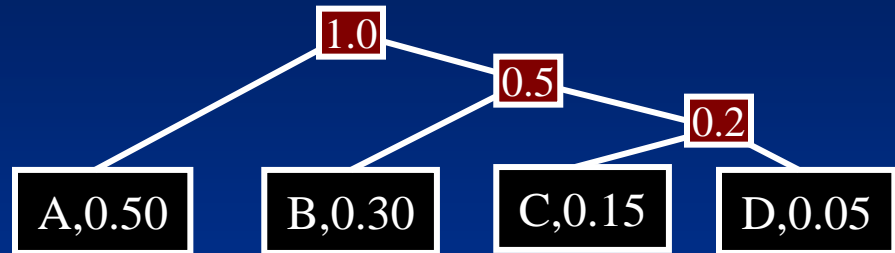
$$Entropy = - \sum_i P_i \log_2(P_i)$$

- A **Huffman Code** is a frequency dependent code that can be used to approximate the theoretical lower bound (entropy) on the average number of bits required per symbol.

# Huffman Code

1. Write down the symbols and their probabilities. They are the terminal nodes.

- A, 0.50
- B, 0.30
- C, 0.15
- D, 0.05



2. Find and mark the two smallest nodes. Add a node w/ arcs to the nodes marked.
3. Set the probability of the new node to the sum of marked nodes.
4. Repeat steps 2 and 3 until all nodes have been marked, except one - the root.
5. The encoding is found by tracing the path from the root to the symbol, w/ left = 0, right = 1.

**A ~ 0, B ~ 10, C ~ 110, D ~ 111**

# Example (cont.)

- Example

- A, 0.50, A~0
- B, 0.30, B~10
- C, 0.15, C~110
- D, 0.05, D~111

- Entropy =  $-(0.5 \log(0.5) + 0.3 \log(0.3) + \dots + 0.05 \log(0.05))$   
= 1.6477 bits per symbol
- Actual =  $(0.5 * 1 + 0.3 * 2 + 0.15 * 3 + 0.05 * 3)$   
=  $0.5 + 0.6 + 0.45 + 0.15$   
= 1.7 bits per symbol



# Huffman Code Example

**Question: Compute the Huffman Code**

<b>Symbol</b>	<b>Frequency</b>	<b>Huffman Code</b>
<b>A</b>	<b>0.6</b>	
<b>B</b>	<b>0.3</b>	
<b>C</b>	<b>0.1</b>	

**What is the average number of bits required per symbol using a Huffman Code?**

**Note: The theoretical lower bound on the number of bits required per symbol is 1.295461844 .**

# Context Dependent Code

- Use **contextual information** to develop a code.
- Example: If the same symbol occurs a large number of times, use **run-length encoding**.
- For example, a typical FAX has a large amount of **blank space**; run-length encoding can be used to encode binary bit strings containing mostly zeros (or blanks).
- Each k-bit symbol tells how many '0' bits (or blanks) occurred between consecutive 1-bits, a symbol w/ all 1-bits means that the total distance is  $(2^k - 1)$  plus the value of the next symbol(s).



# Example

- Let  $k=3$
- Then, the data:

0001000001000000100000000000000010000001

3

5

6

14

6

is encoded as 011 101 110 111 111 000 110.

3

5

6

14

6



# Summary

- **Topics**

- Physical Layer - transmission of raw bits using hardware.
- Data Link Layer - reliable transmission between directly connected machines.
- Network Layer - transmission between any two hosts (routing, congestion control, flow control).
- Transport Layer – reliable end-to-end tx between any two hosts.
- Session Layer – provide session support.
- Presentation Layer – present data in a meaningful format.

- **Assignments**

- Homework #3 – 10/25/2002
- Homework #4 – 10/28/2002
- Quiz #2 – 11/1/2002
- Read Ch. 7

