

CS241 System Programming Interprocess Communication (VII)

Klara Nahrstedt

Lecture 40

4/28/2006

Contents

- Communication Paradigms – Putting it Together
- Remote Procedure Call
- Performance Issues – Things to be aware off

Administrative

- Read R&R Chapter 19,20
- MP5 is on – deadline May 1 (Monday), 4pm
- **Conflict Exam – Check the newsgroup posting**
- Homework 2 – posted April 24 – deadline May 3, midnight
- **Quiz 12 – May 3 → OPTIONAL !!!**
 - Cover Synchronization
 - Cover File Systems
 - Cover Memory Management
- There is more than 1 metaserver running for mp5, all the students appear to be using just one of them, check the status page, and spread out a little better; similarly, if need be, more servers will be running over the weekend

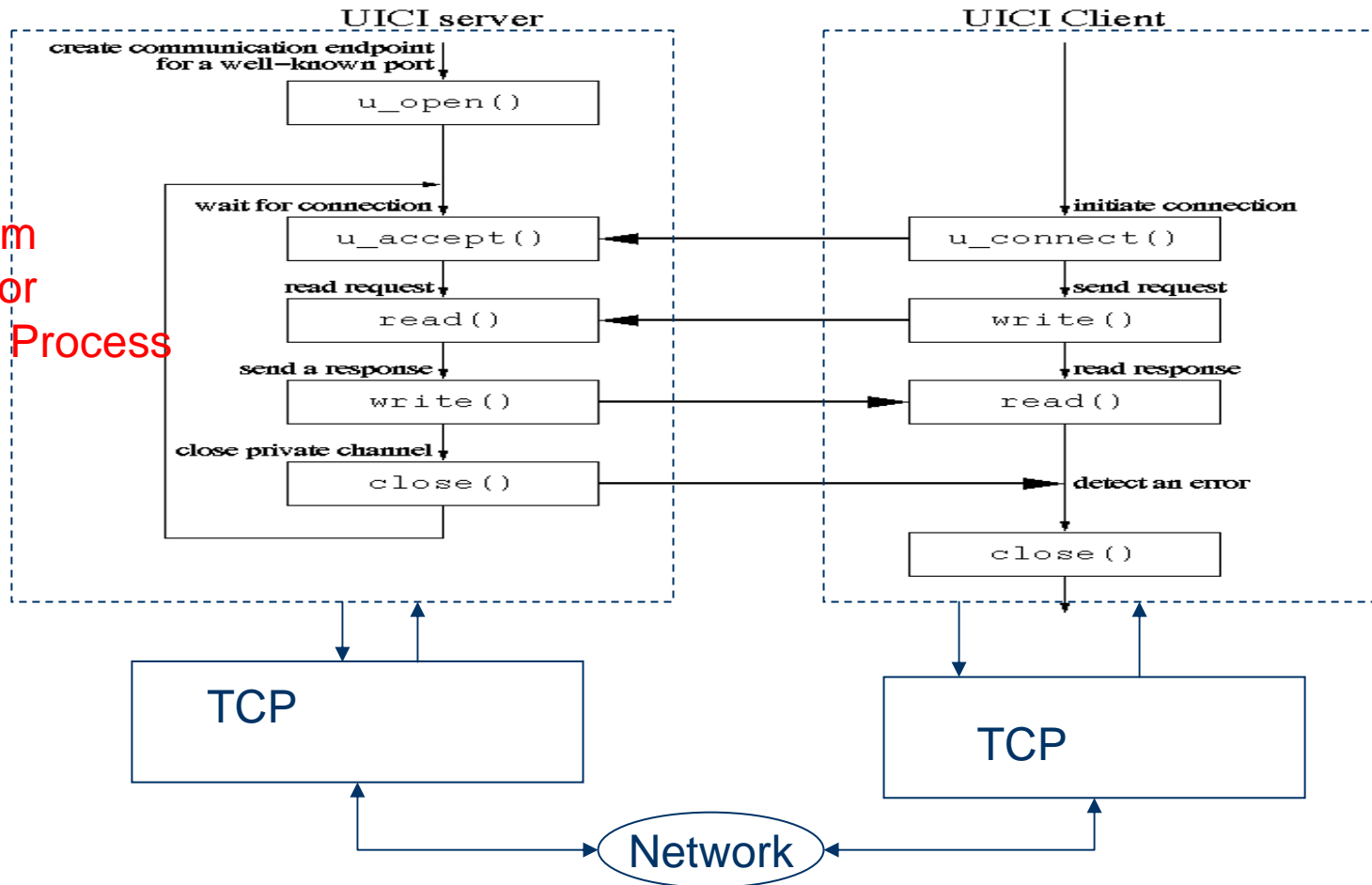
Interprocess Communication Paradigms within a Single Machine

- **Pipes**
 - Pipe command creates a communication buffer that two processes (Parent and Child) can use to access via `files[0]` to read from and `files[1]` to write to.
- **Shared memory**
 - `Shmid` – shared memory segment is attached to the calling process
 - `Shmget`, `Shmat` assist in creating the shared memory segment and attaching it to the calling process
 - `Read/write` assist in communicating via shared memory
- **Message Queues**
 - `Msgqid` – major data structure for message queue is needed
 - `Msgget`, `msgsnd`, `msgrsv` assist in creating the message queue and communicating via message queue

Connection-Oriented Communication Paradigm for Client-Server Interaction

Program Logic for Server Process

Program Logic for Client Process



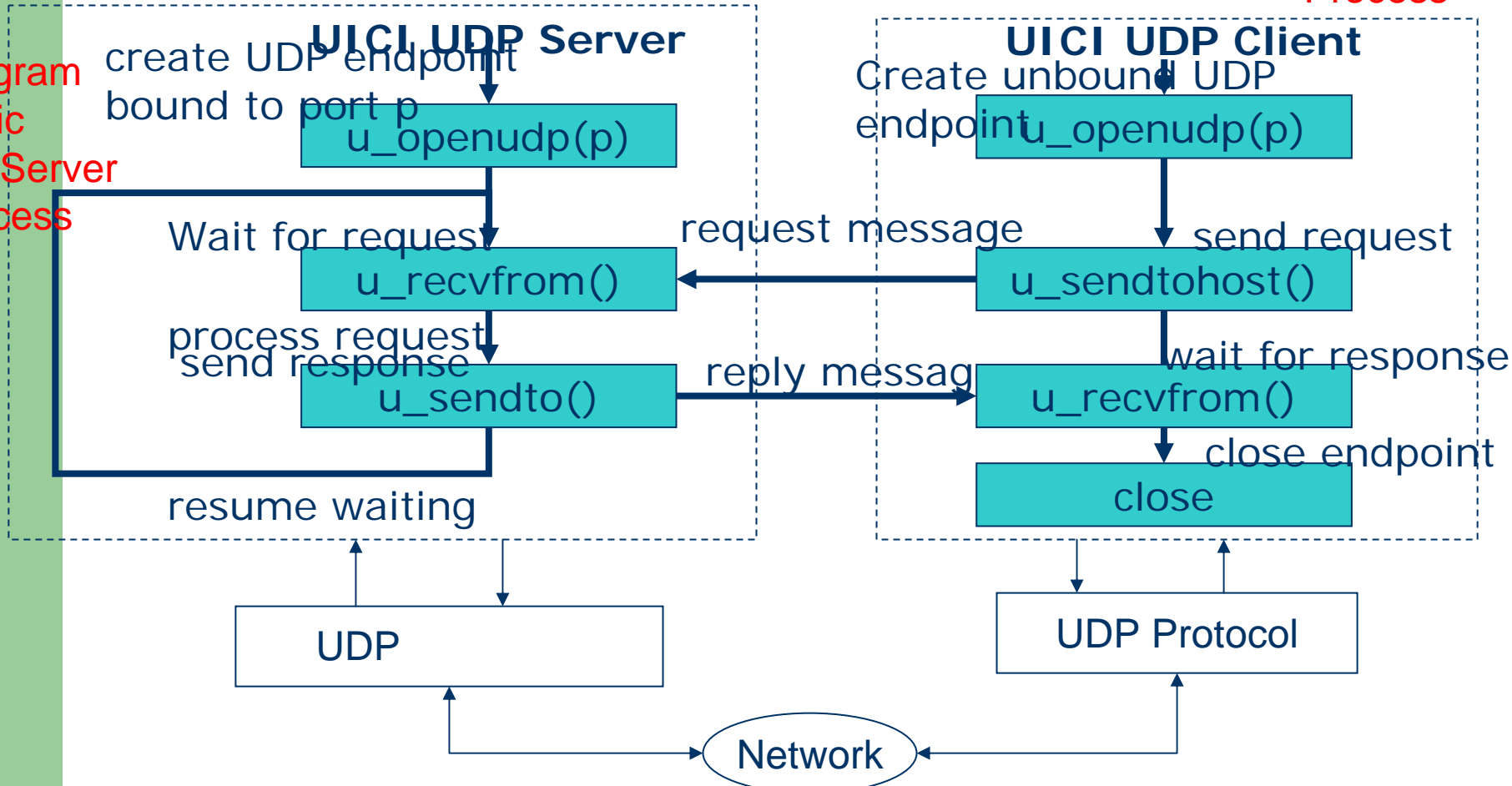
Connection-Oriented Communication Paradigm between two Remote Machines

- **Connection is established**
 - Setup the socket structure
 - Bind the process and socket to a specific port on each side
 - Use the socket structure and port throughout the whole session until the connection is closed
- **TCP Protocol** ensures that the connection is reliable, packets arrive in order, and connection is on throughout the whole session
- In this paradigm we **don't need connection-related control operations** (e.g., acknowledgement, reply messages) since TCP executes them and controls the data stream.

Connection-less Communication Paradigm between Two Different Machines

Program Logic For Client Process

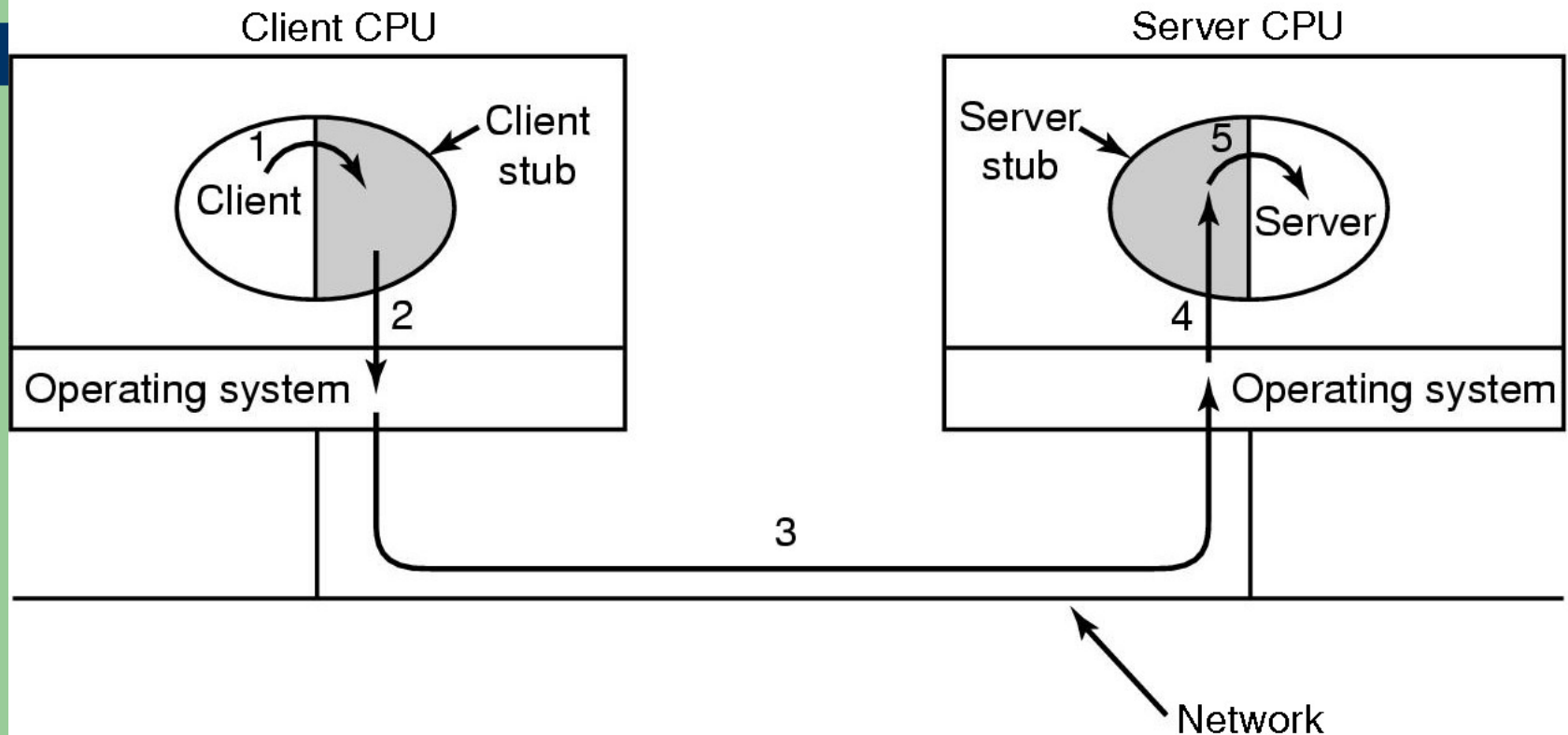
Program Logic For Server Process



Connection-less Communication Paradigms between Two Different Machines

- **No connection setup**
- Datagram messages are being exchanged
- Since no connection exists, each message must have in its send/rcv message the source and destination information.
- Underlying UDP protocol provides only a **forwarding** mechanism, nothing else
- If the application processes want more reliability, but want to run on top of the UDP protocol, they must provide this capability themselves via 'request-reply' protocol or 'request-reply-acknowledgement' protocols

Remote Procedure Call (1)



- Steps in making a remote procedure call
 - the stubs are shaded gray

Remote Procedure Call (2)

Implementation Issues

- Cannot pass pointers
 - call by reference becomes copy-restore (but might fail)
- Weakly typed languages
 - client stub cannot determine size
- Not always possible to determine parameter types
- Cannot use global variables
 - may get moved to remote machine

Remote Procedure Call (3)

- Parameters of RPC are **marshaled** into message in the RPC client process
- Message sent and local RPC process waits
- Remote machine receives RPC message, spawns remote process that will execute the remote computation on behalf of the user client
- Remote process assumes same protection domain as local process

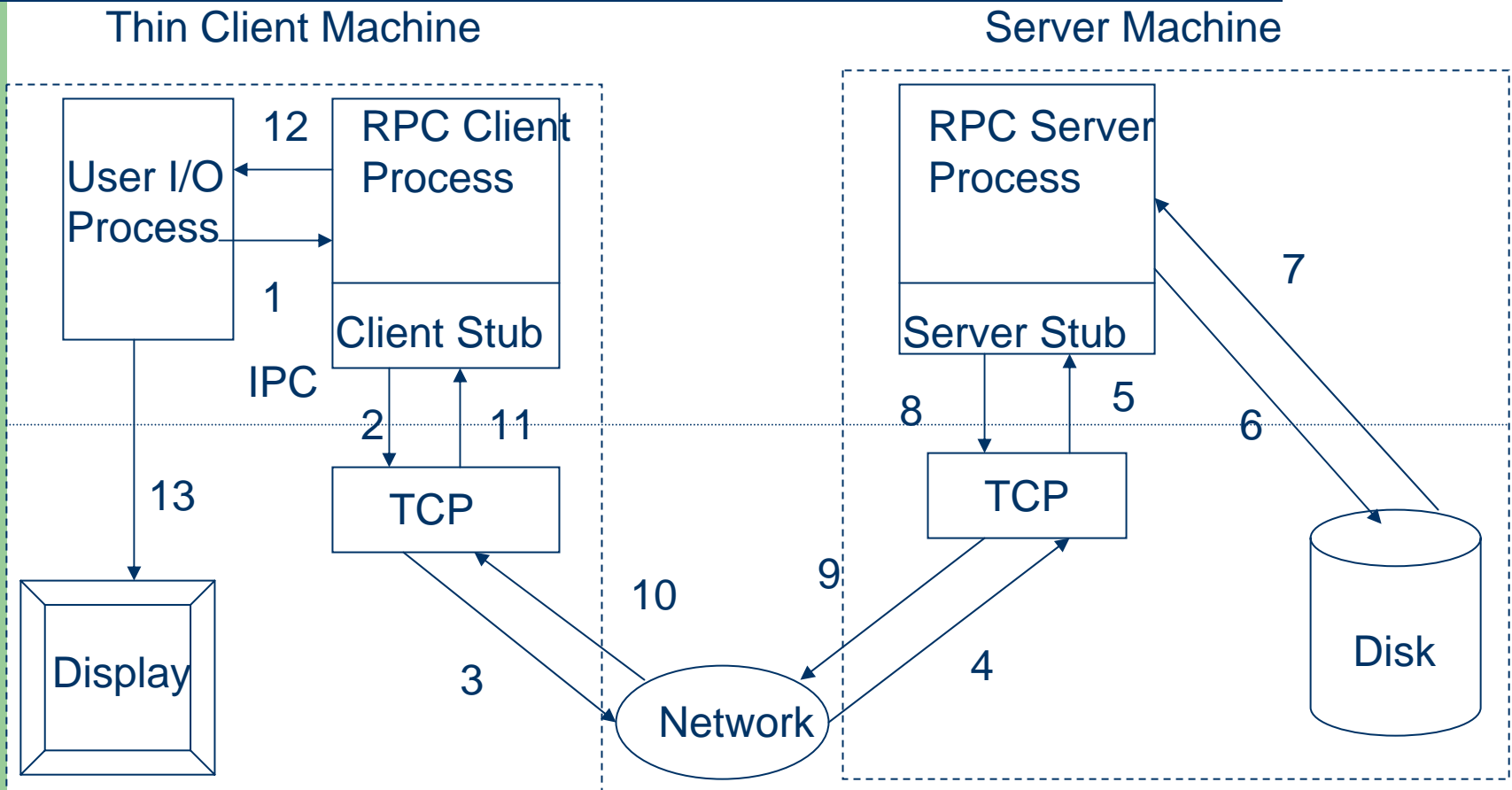
More RPC (4)

- Remote process **assembles parameters** and makes procedure call
- Remote process marshals return parameters into message
- Message sent and remote process dies
- Local process resumes and unpacks result

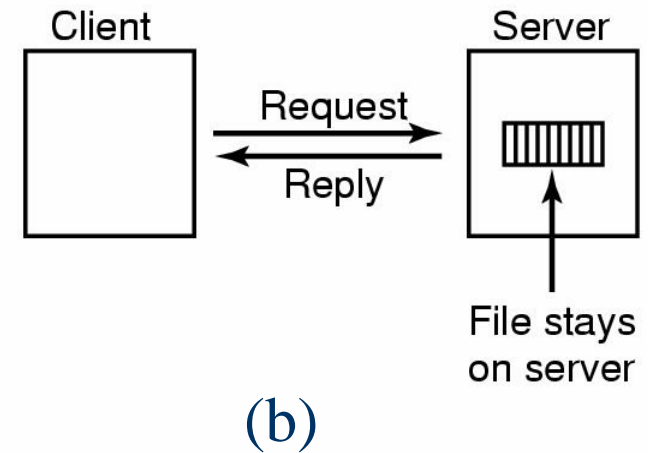
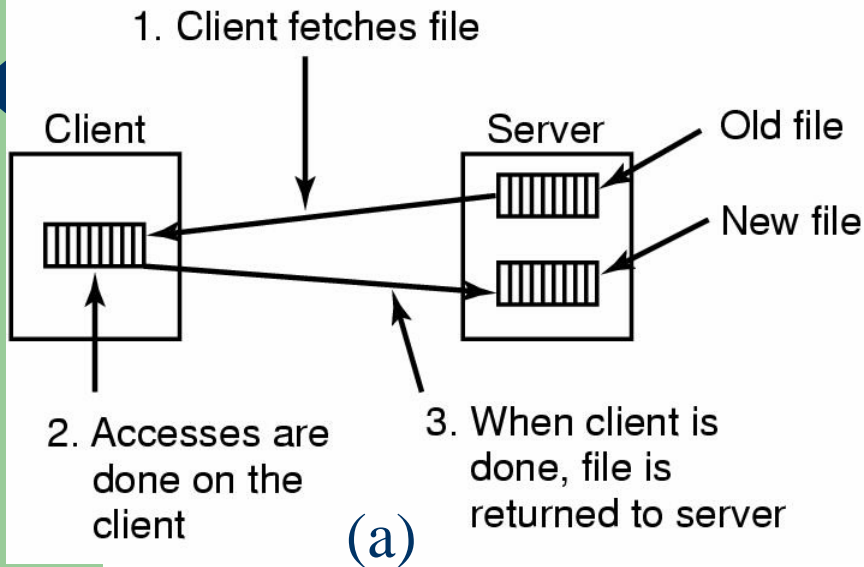
RPC Semantics (5)

- at most once
- at least once
- once
- idempotent

Application of RPC



Application of RPC (Distributed File System)



Transfer Models

(a) upload/download model

(b) remote access model

Performance Issues when Exchanging Messages between Clients and Servers

- Synchronous or asynchronous
- Loss, Timeouts, Acknowledgments, Sequence numbers
- Congestion

Synchronous or Asynchronous

- Synchronous
 - Process waits for message – Rendezvous
 - Process waits for message to be received
- Asynchronous
 - Pool of messages with query mechanism to retrieve a message
 - Processes send messages without delay
 - Processes receive messages from pool

Loss

- Timeouts
 - Try again after some time limit
 - Heart Beat or Keep Alive
- Acknowledgments
 - Reply after message
 - Lost acknowledgements
- Sequence numbers
 - Losses become known

Naming

- Authentication
 - Who is the message from?
 - Who gets the message?
- Ports
 - Send to a Port ID – any server process can receive.
- Process IDs – only one process receives

Other Performance Issues

- How many times is a message copied before its delivered?
- Loss of cache by receiving message
- Wait time for synchronous/asynchronous messages
- Search time for asynchronous messages
- How do network functions impact message passing/exchange
 - Network Model – Topology
 - Routing
 - Contention

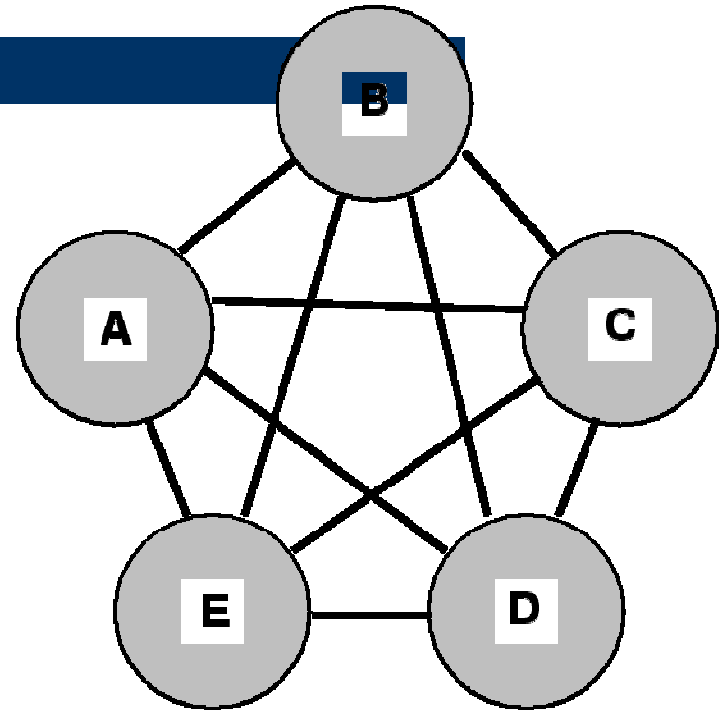
Network Model - Topology: criteria

- Basic cost
 - Expense.
- Communication cost
 - Latency.
- Reliability
 - What happens when a link or node fail.

Fully Connected

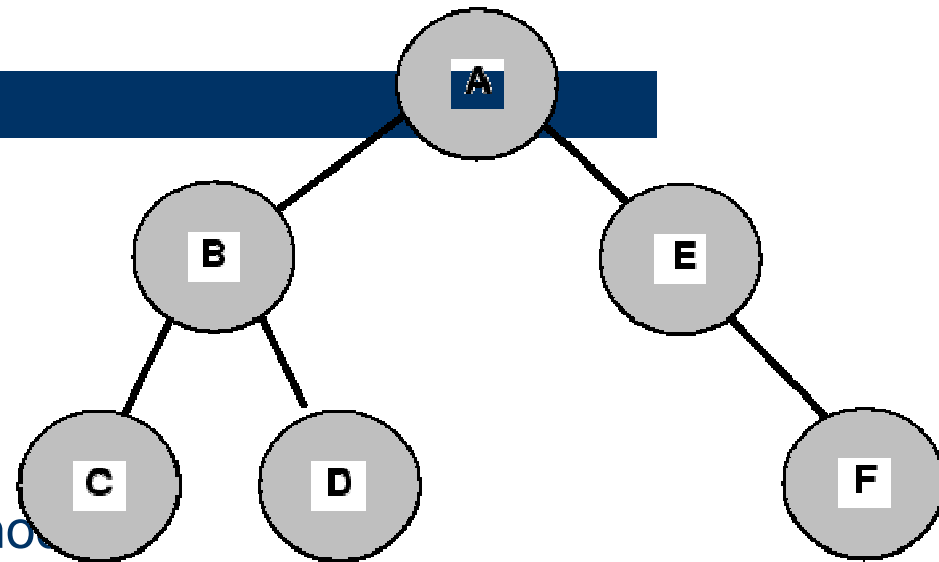
$$\frac{n \times (n-1)}{2}$$

- Basic cost:
 - high links
- Communication cost:
 - low (single hop)
- Reliable
 - not likely to partition



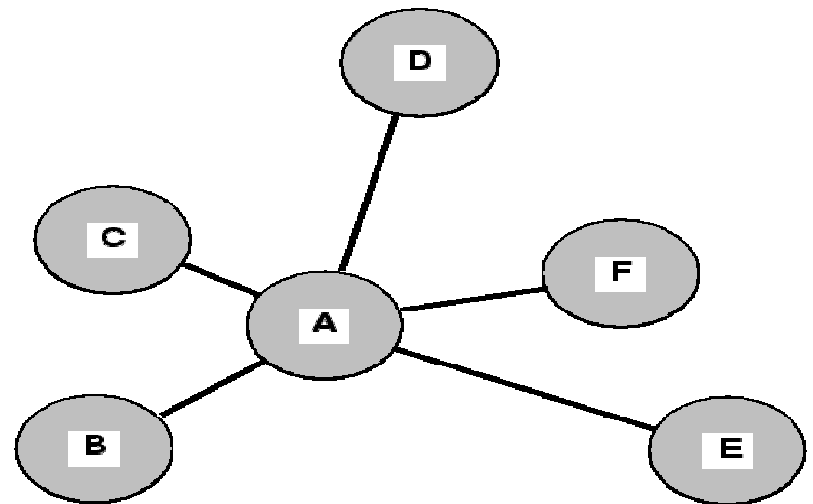
Hierarchical

- Basic cost:
 - Low, $n-1$ links
- Communication cost:
 - multiple hops
 - but organized to accommodate likely traffic flows (subtrees represent localities)
 - Root may become bottleneck.
- Reliable
 - single failure likely to partition



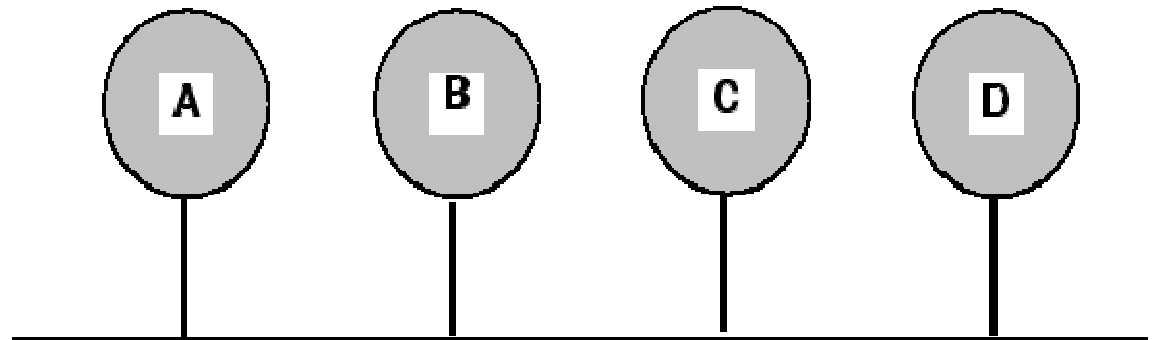
Star-Connected Network

- Basic cost:
 - ?
- Communication cost:
 - ?
- Reliable
 - ?



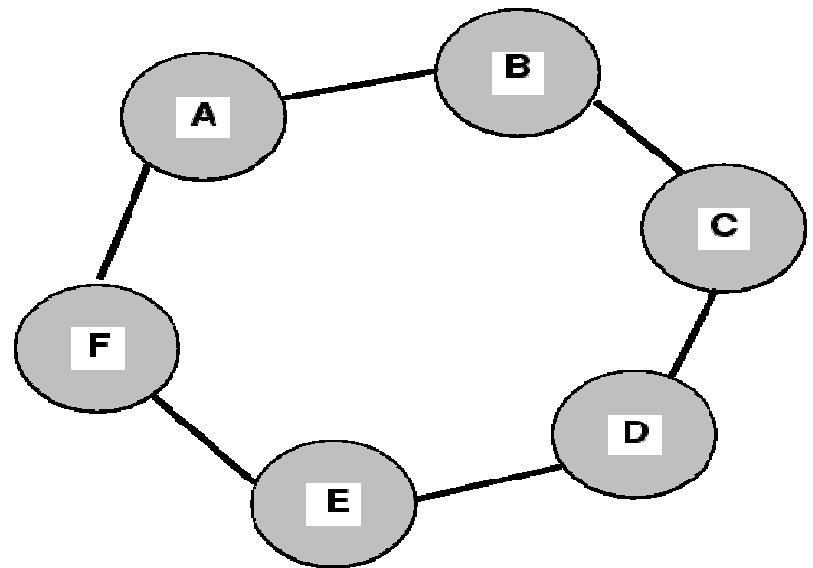
Bus-Connected Network

- Basic cost:
 - ?
- Communication cost:
 - ?
- Reliable
 - ?



Ring-Connected Network

- Basic cost:
 - ?
- Communication cost:
 - ?
- Reliable
 - ?



Routing

- Sends message with network address to correct physical network address
- **Fixed Routing**
 - Path is set up from one point to another and doesn't change unless a hardware failure occurs.
- **Virtual Circuit**
 - Path is fixed for one communication session.
- **Dynamic Routing**
 - Each message may go a different route.

Contention - Collision Detect

- Before transmission, **listen to network** for free link.
- During transmission, listen to make sure that there is not a simultaneous transmission.
- When collision occurs, use **back off** strategy to avoid busy wait.
- Wait for **random number of time units**.
- Wait for **exponential amount of time** based on number of attempts
- Doesn't prevent indefinite wait to transmit.

Contention - Token Passing

- **Pass token around network** -- special message.
- When receive token, can transmit one message but must then pass on the token.
- Provides fair message transmission.
- Lost tokens must be replaced. Use time out.
- Use election algorithm to choose a node to create a token.

Contention - Message Slots

- A number of **empty messages continuously circulate** around the network.
- Node grabs empty slot, fills it with message.
- Receiving node removes message and replaces empty message.

Summary

- Be careful when choosing a **communication paradigm** – depends on application you want to implement
- Be aware of the **underlying Internet transport protocol** (TCP or UDP) what kind of functions it provides to the user processes
- Be aware of the underlying network, i.e., its topology, routing and contention mechanisms and algorithms the low level networking elements use