Communication

Distributed Systems IT332

Outline

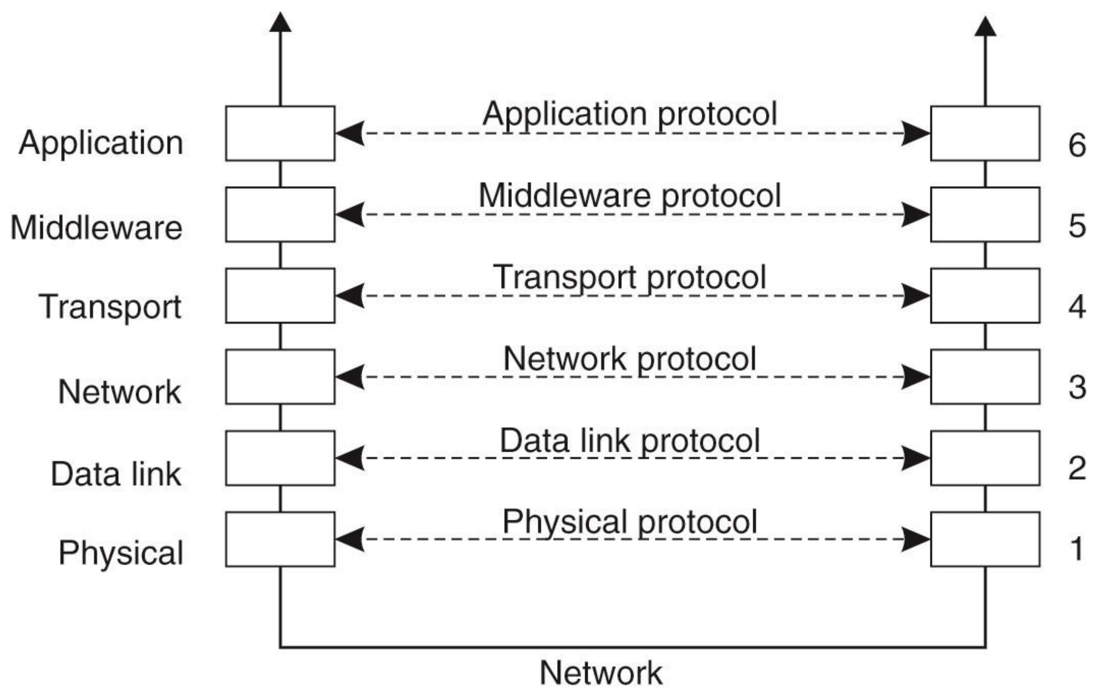
- Fundamentals
  - Layered network communication protocols
  - Types of communication
- Remote Procedure Call
- Message-Oriented Communication
- Multicast Communication

Layered Network Communication Protocols

- Low-level layers
  - Physical layer: transmitting bits between sender and receiver
  - Data link layer: transmitting frames over a link, error detection and correction
  - Network layer: routing of packets between source host and destination host
    - IP – Internet's network layer protocol
- Transport layer: process-to-process communication
  - TCP and UDP -Internet's transport layer protocols
    - T CP: connection-oriented, reliable communication
    - UDP: connectionless, unreliable communication
- Higher-level layers
  - Session and presentation layers are not present in the Internet protocol suite
  - Application layer contains applications and their protocols
    - E.g.,Web and HTTP, Email and SMTP

Middleware Layer

- Middleware provides common services and protocols that can be used by many different applications
  - High-level communication services, e.g., RPC, multicasting
  - Security protocols, e.g., authentication protocols, authorization protocols
  - Distributed locking protocols for mutual exclusion
  - Distributed commit protocols



Types of Communication

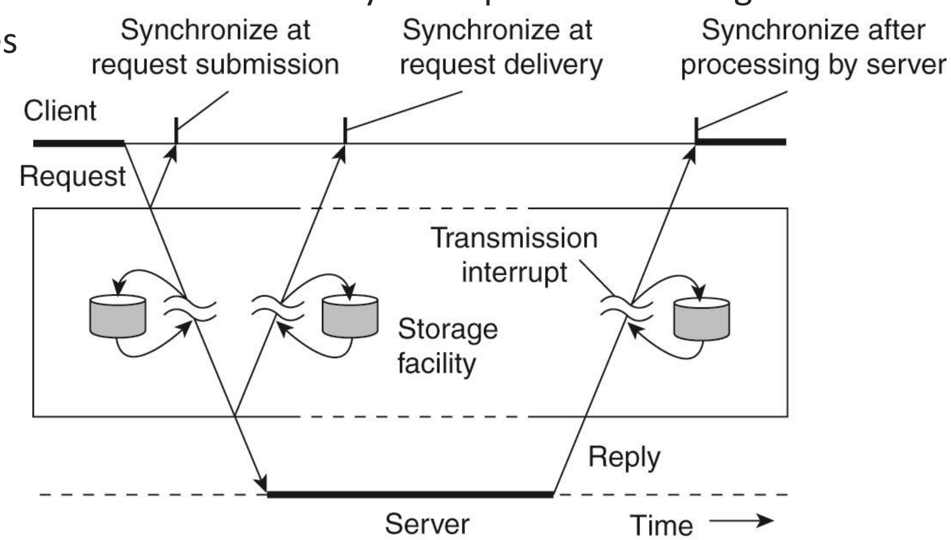
- Transient vs. persistent communication
- Synchronous vs. asynchronous communication

Transient vs Persistent Communication

- Transient communication: middleware discards a message if it cannot be delivered to receiver immediately (sender and receiver run at the same time based on request/response protocol, the message is expected otherwise it will be discarded)
  - Example: applications using TCP (FTP),UDP(Video streaming)
- Persistent communication: messages are stored by middleware until receiver can accept it
  - Receiving application need not be executing when the message is submitted.
  - Example: Email

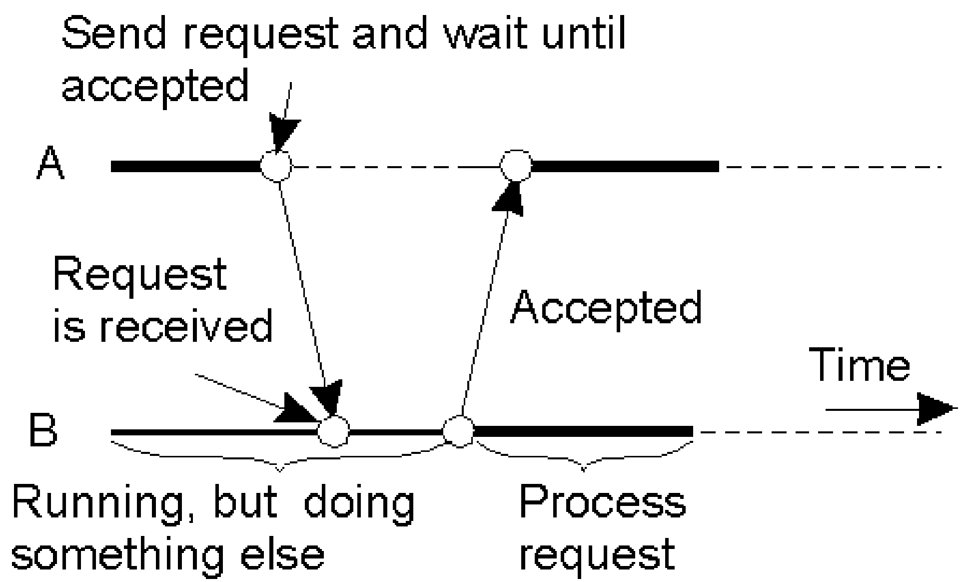
Synchronous vs Asynchronous Communication

- Synchronous communication: sender blocks until its request is known to be accepted
  - Sender and receiver must be active at the same time
  - Sender execution is continued only if the previous message is received and processed.
  - Three places



- Asynchronous communication: sender continues execution immediately after sending a message
  - Message stored by middleware upon submission
  - Message may be processed later at receiver's convenience

Activity

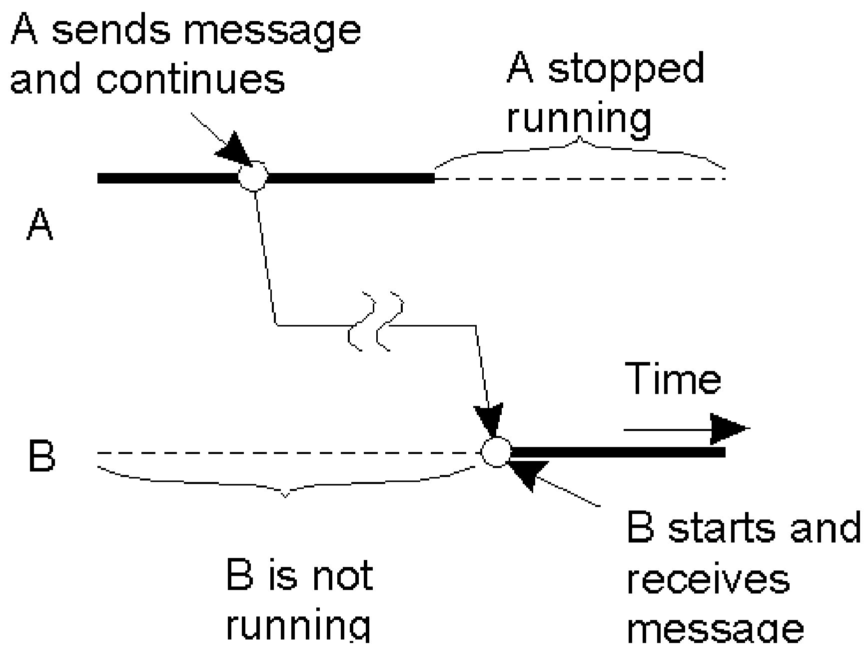


A. Persistent asynchronous communication

B. Persistent synchronous communication

C. Transient asynchronous communication

D. Transient synchronous communication

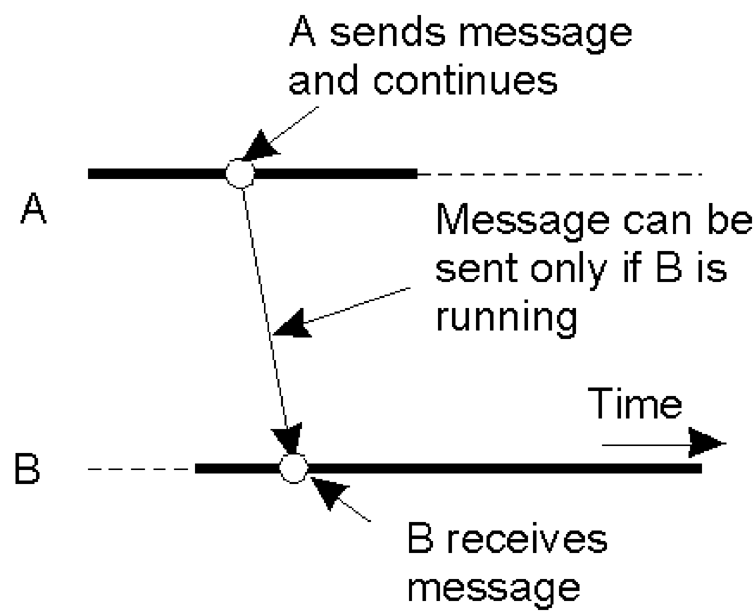


A. Persistent asynchronous communication

B. Persistent synchronous communication

C. Transient asynchronous communication

D. Transient synchronous communication

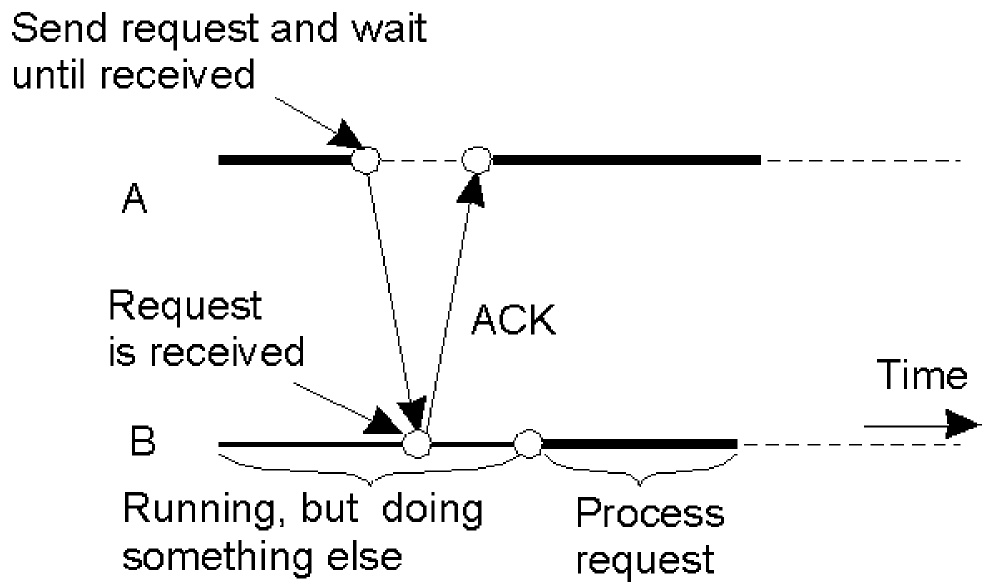


A. Persistent asynchronous communication

B. Persistent synchronous communication

C. Transient asynchronous communication

D. Transient synchronous communication



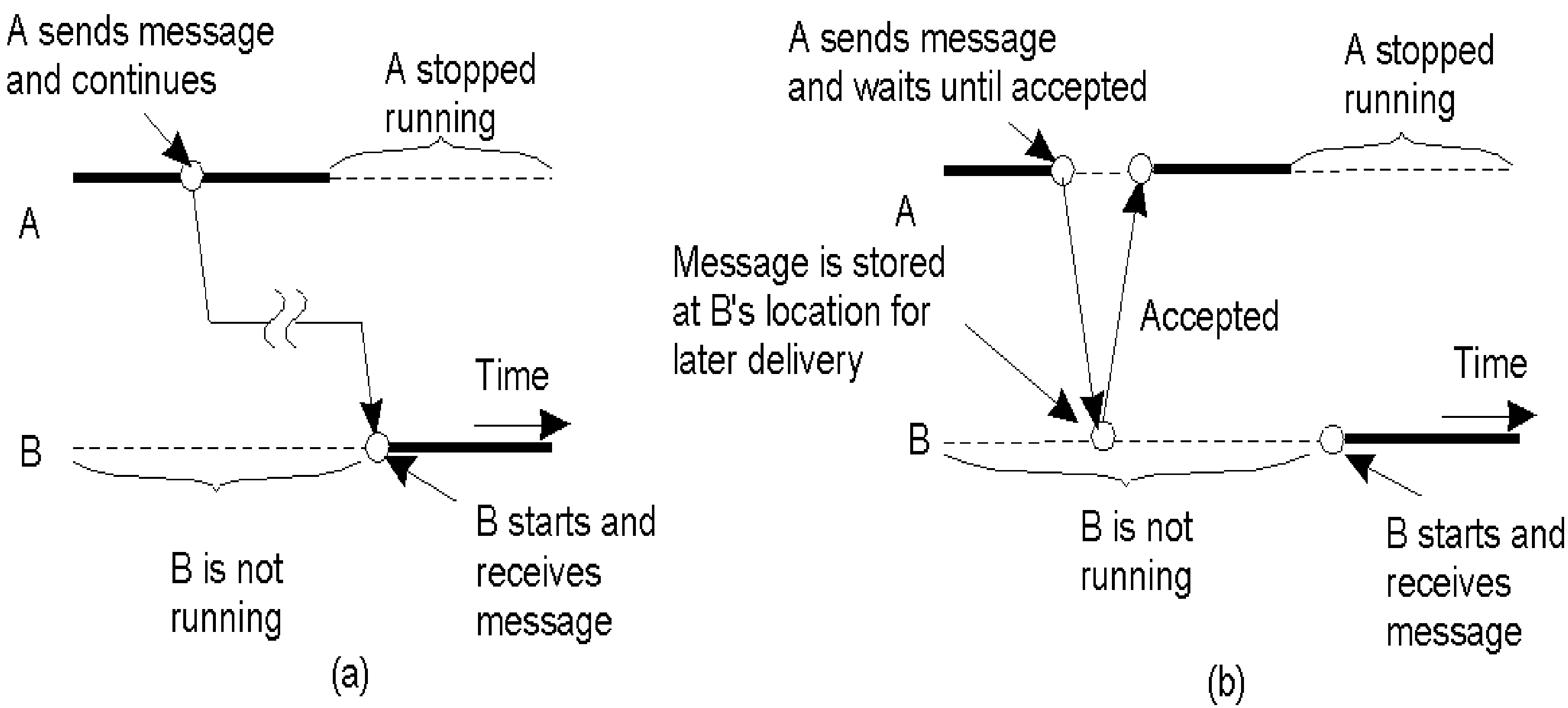
A. Persistent asynchronous communication

B. Persistent synchronous communication

C. Transient asynchronous communication

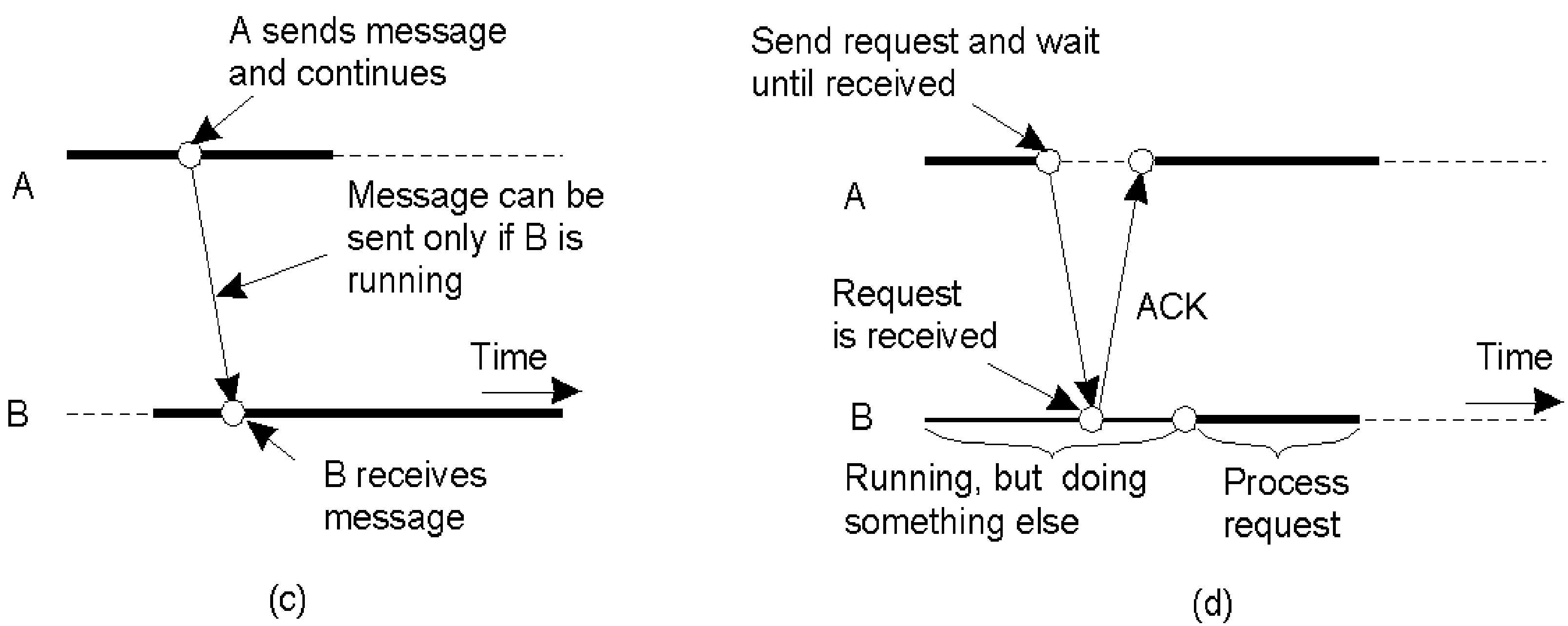
D. Transient synchronous communication

Persistence and Synchronization Combinations



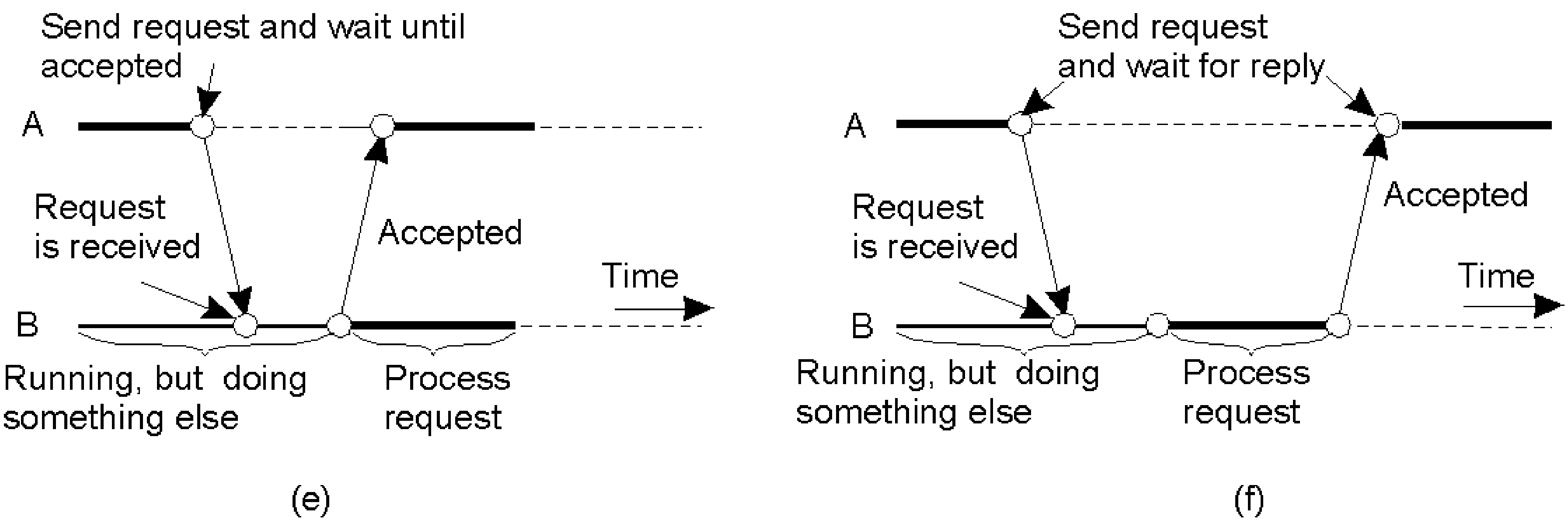
a) Persistent asynchronous communication (e.g. Email)
b) Persistent synchronous communication

Persistence and Synchronization Combinations



c) Transient asynchronous communication (one way RPC, e.g. UDP)
d) Receipt-based transient synchronous communication

Persistence and Synchronization Combinations



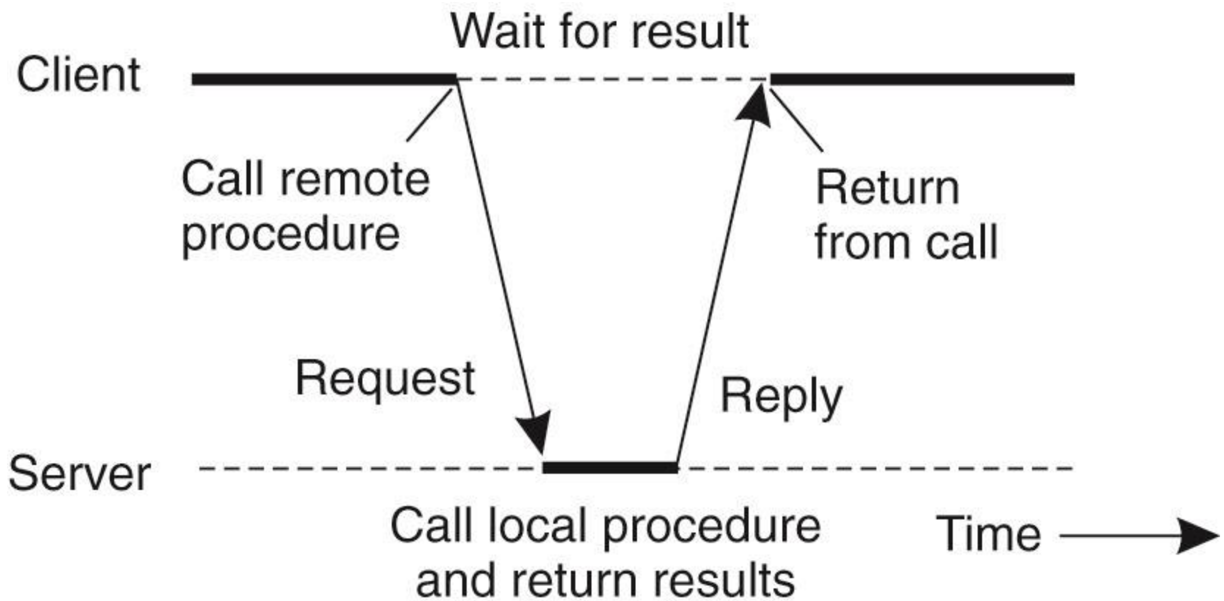
e) Delivery-based transient synchronous communication at message delivery(asynchronous RPCs)

f) Response-based transient synchronous communication (RPCs)

Remote Procedure Call (RPC)

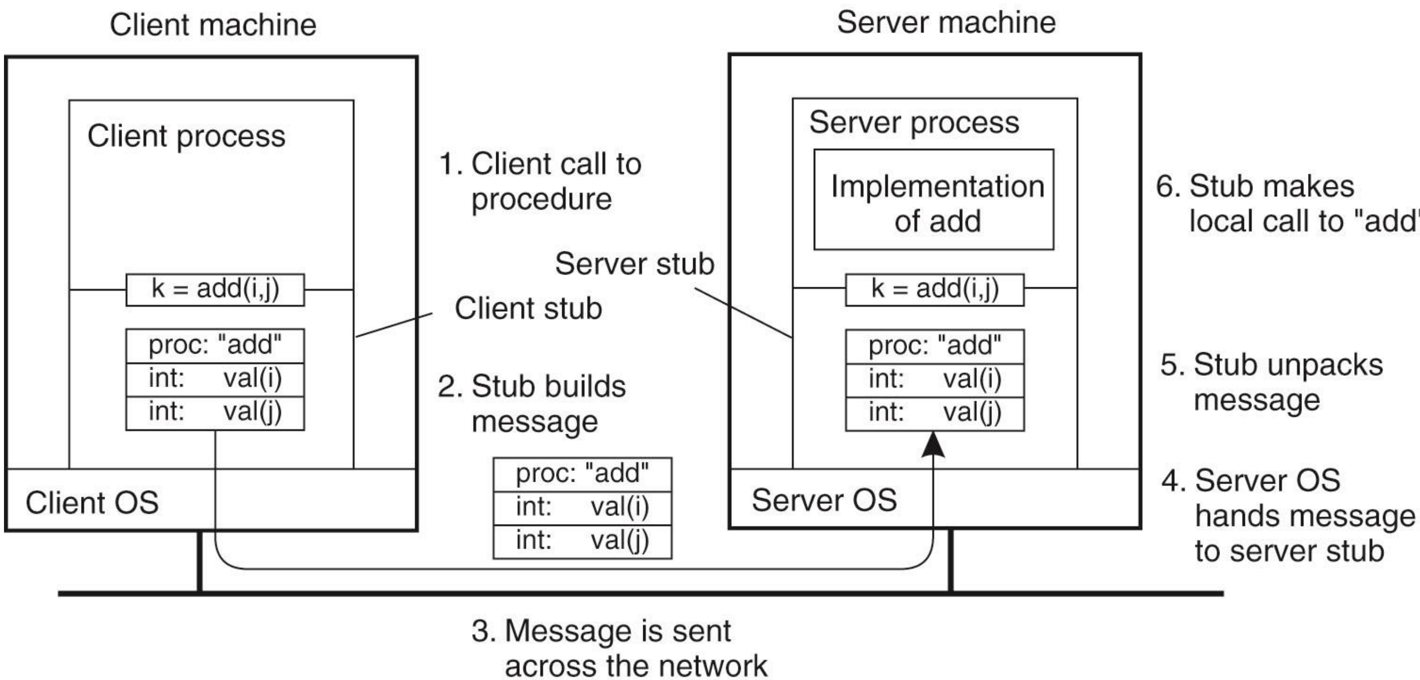
- Replace the explicit message passing model with the model of executing a procedure call on a remote node
  - Synchronous - based on blocking messages
  - Message-passing details hidden from application
  - Procedure call parameters used to transmit data
  - Client calls local “stub” which does messaging and marshaling
- Example RPC frameworks: SUNRPC,DCE RPC, XML-RPC, SOAP

RPC



RPC between a client and a server

Basic RPC Operation



1. Client program calls client stub.
2. Client stub packs parameters into message (marshaling), calls local OS.
3. Client's OS sends message to remote OS.
4. Remote OS delivers message to server stub.
5. Server stub unpacks message, calls server procedure.
6. Server does work, returns result to stub.
7. Server stub packs result in message, calls local OS.
8. Server's OS sends message to client's OS.
9. Client's OS delivers message to client stub
10. Client stub unpacks result (unmarshalling), returns to client program.

Parameter Marshalling

- Client/server stub must pack ("marshal") parameters/result into message structure
- May have to perform other conversions when processes on heterogeneous architectures communicate:
  - Byte order (big endian vs little endian)
  - Dealing with pointers
  - Convert everything to standard ("network")format, or
  - Message indicates format, receiver converts if necessary

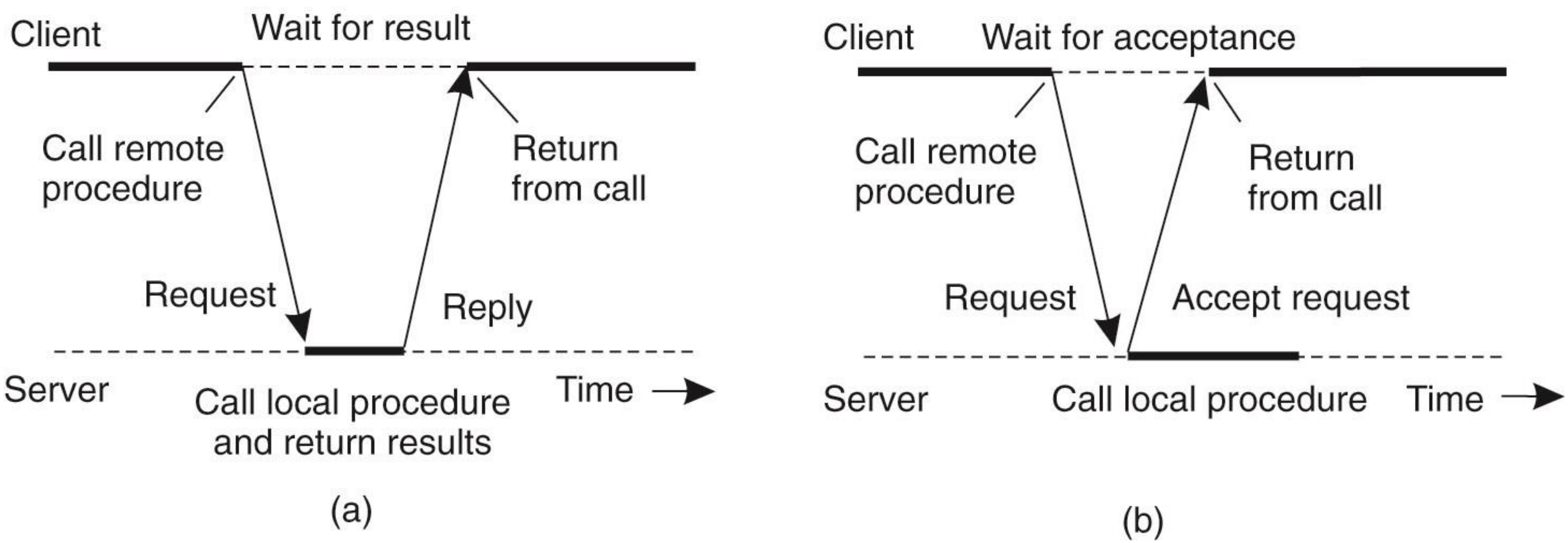
Parameter Marshalling

- Message data must be pointer free
- Cannot pass pointers
  - all by reference becomes copy-restore

Other RPC Models

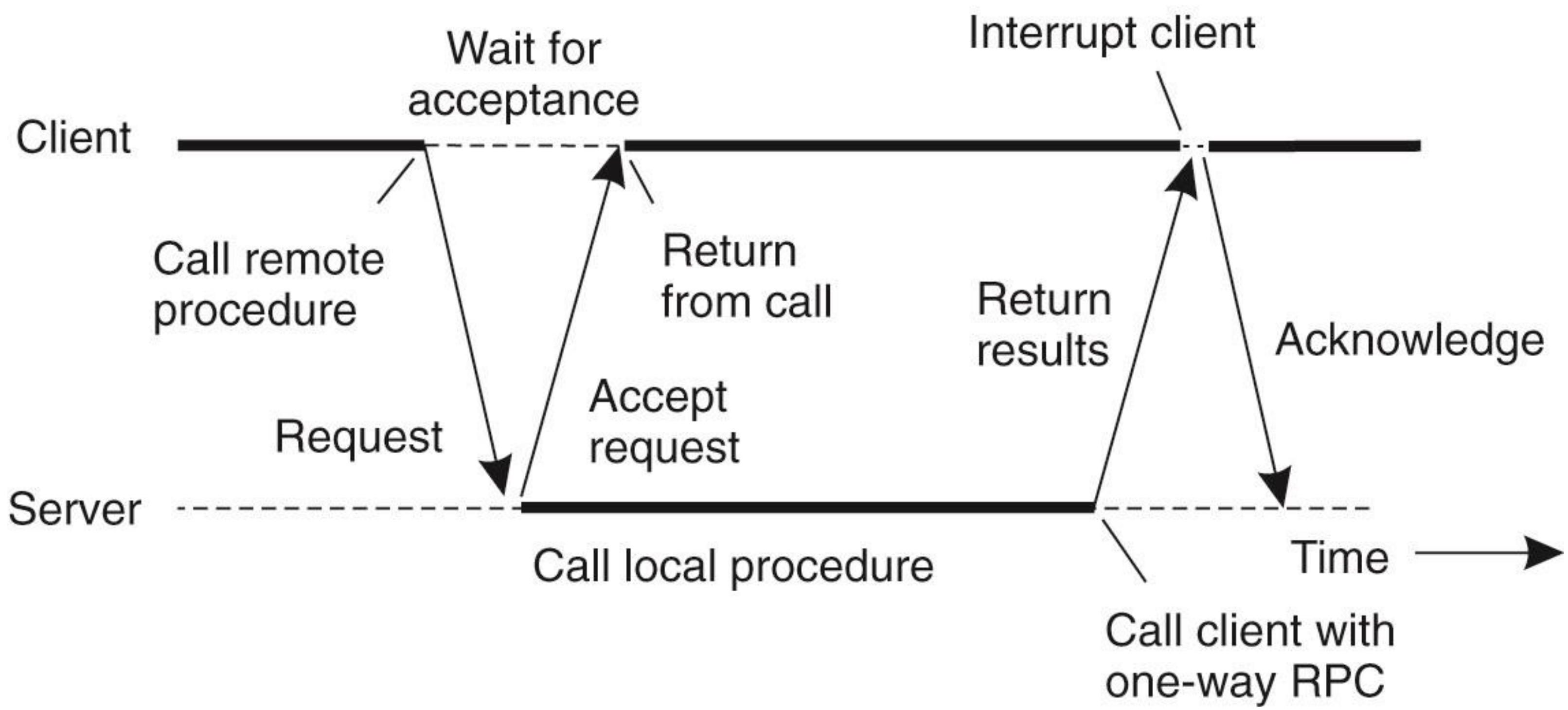
- Conventional RPC: client blocks until a reply is returned
- Asynchronous RPC
  - No need to wait for a reply when there is no result to return
  - RPC returns as soon as the server acknowledges receipt of the message
- Deferred synchronous RPC
  - Client needs a reply but reply isn't needed immediately
  - Use two asynchronous RPCs: server sends reply via another asynchronous RPC
- One-way RPC (a variant of asynchronous RPC)
  - Client does not even wait for an ACK from the server
  - Limitation: client cannot know for sure whether its request will be processed.

Asynchronous RPC



(a) The interaction between client and server in a traditional RPC.
(b) The interaction using asynchronous RPC

Deferred Synchronous RPC



A client and server interacting through two asynchronous RPCs

Message-Oriented Communication

- Message-Oriented Transient Communication
  - Berkeley sockets
- Message-Oriented Persistent Communication
  - Message-queuing systems

Message-Oriented Transient Communication

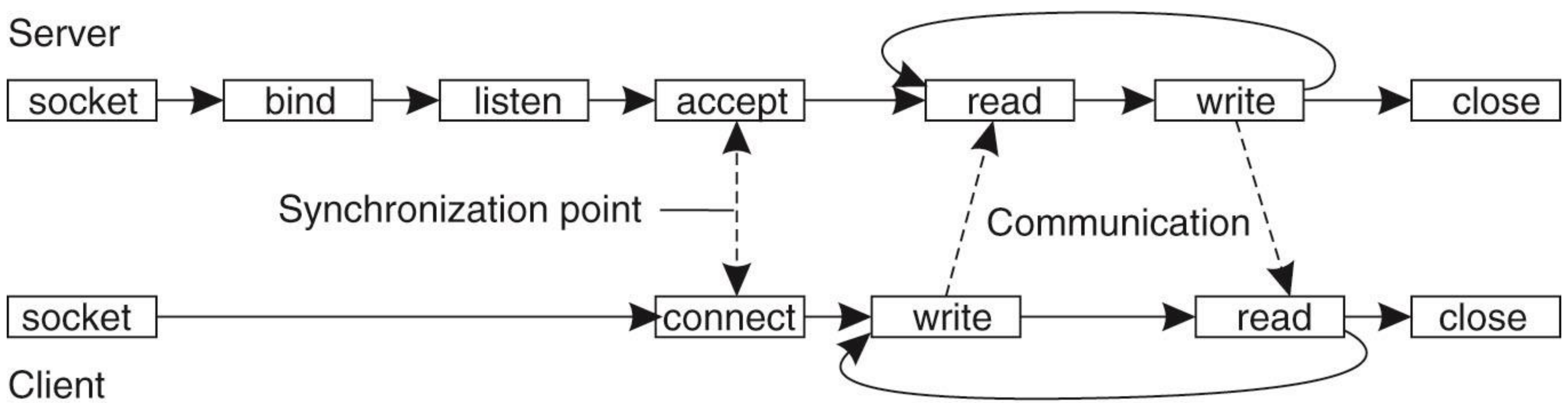
- Many distributed systems/applications are built on top of simple message-oriented model offered by the transport layer
- Berkeley sockets: a standard interface of the transport layer

Berkeley Sockets

Primitive	Meaning
Socket	Create a new communication end point
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

The socket primitives for TCP

Client-Server Communication Using TCP Sockets



Message-Queuing (MQ) Systems

- Also called Message–Oriented Middleware (MOM): example :email system.
- Support asynchronous persistent communication
- Applications communicate by inserting messages in queues
  - Messages can only be added to and retrieved from local queues: senders place messages in source queues, receivers retrieve messages from destination queues
  - Message contains name or address of a destination queue
  - MQ system provides queues to senders and receivers, transfers messages from source queue to destination queue
- Very similar to email but more general purpose (i.e., enables communication between applications and not just people)

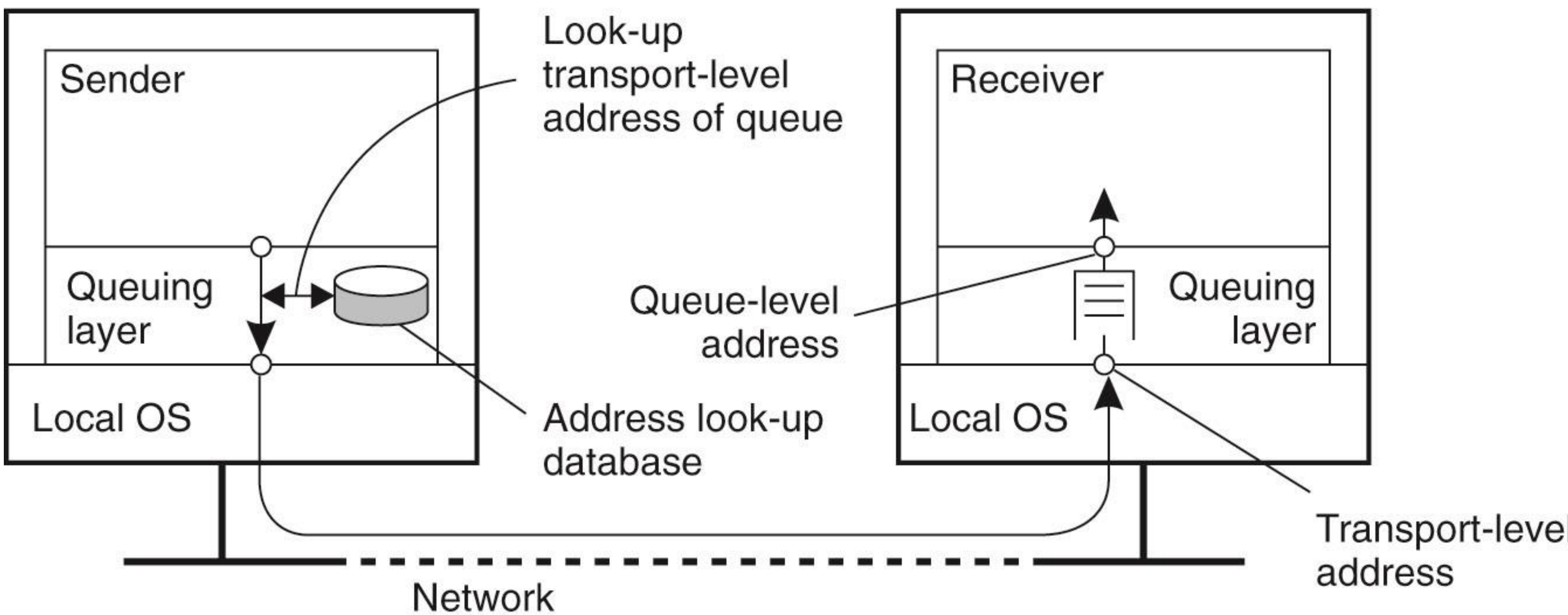
Message-Queuing Systems (continued)

Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block
Notify	Install a handler to be called when a message is put into the specified queue

Basic interface to a queue in a message-queuing system

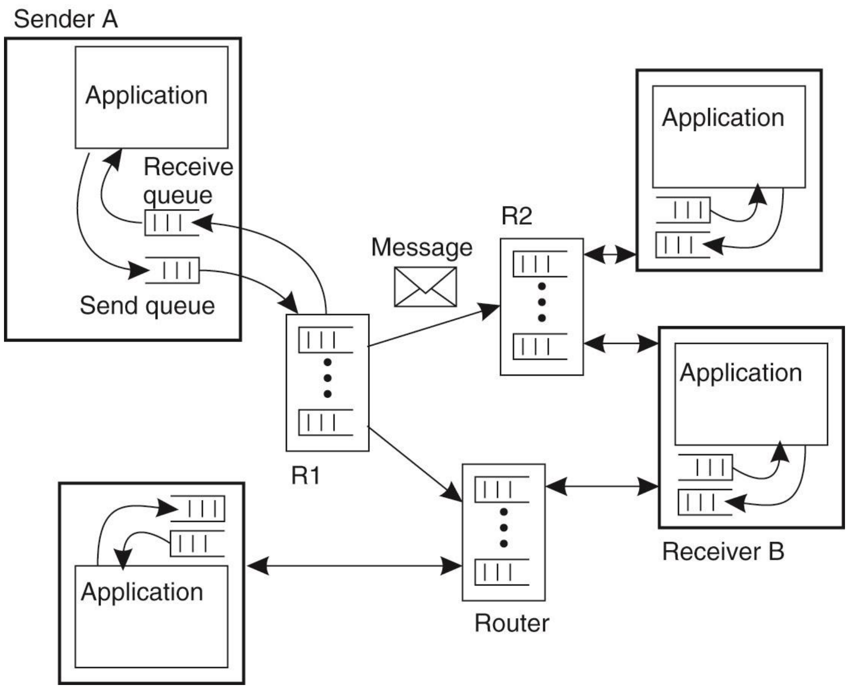
General Architecture of a Message-Queuing System

- Message-queuing system maintains a mapping of queue names to network locations



General Architecture of a Message-Queuing System (continued)

- Having each queue manager maintain a queue-to-location mapping is not scalable
- Routers can help achieve scalability
  - Only routers need be updated when queues are added or removed.
  - Queue manager only needs to know where the nearest router is.



Multicast Communication

- Application-level multicasting
- Gossip-based data dissemination

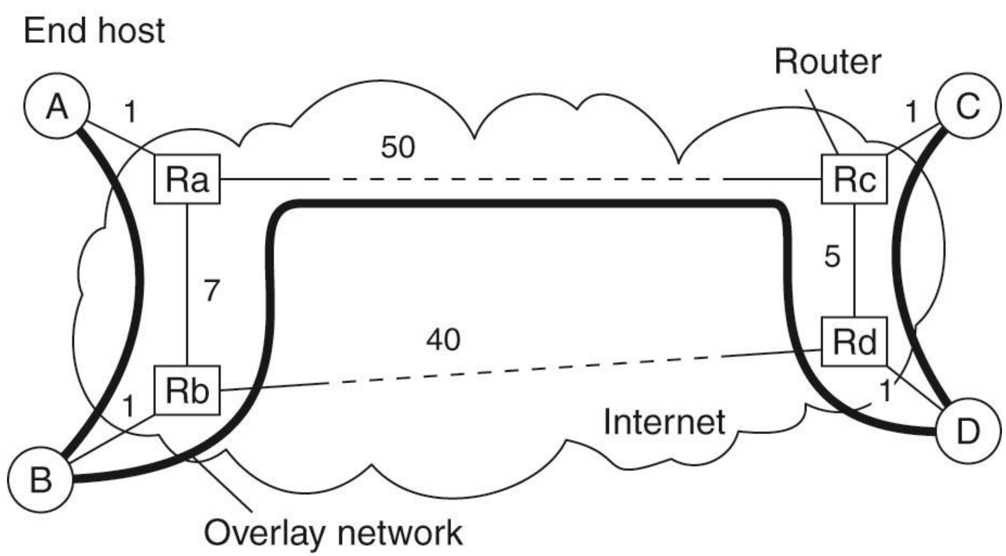
Application-Level Multicasting

- Basic idea: organize nodes of a distributed system into an overlay network and use that network to disseminate data
- Multicast tree construction in Chord
  - Initiator of multicast session generates a multicast identifier mid
  - Lookup succ(mid)
  - Request is routed to succ(mid), which will become the root
  - If P wants to join, it executes Lookup(mid) to send a join request to the root. P becomes a forwarder.
  - When request arrives at Q
    - Q has not seen a join request for mid before → it becomes forwarder; P becomes child of Q. .Join request continues to be forwarded.
    - Q is already a forwarder for mid → P becomes child of Q. No need to forward join request anymore.

Application-Level Multicasting

- Metrics to measure the quality of a multicast tree:
  - **Link stress:** how often does a multicast message cross the same physical link?

    Example: message sent by A needs to cross <Ra, Rb> twice
  - **Stretch:** ratio in delay between overlay path and network-level path. Example: message from B to C follow path of length 71 in overlay but 47 at network-level => stretch = 71/47=1.51.
  - **Tree cost:** total cost of links in the tree



Gossip-Based Data Dissemination

- Use epidemic algorithm to rapidly propagate information among a large collection of nodes with no central coordinator
  - Assume all updates for a specific data item are initiated at a single node
  - Upon an update, try to "infect" other nodes as quickly as possible
  - Pair-wise exchange of updates (like pair-wise spreading of a disease)
  - Eventually, each update should reach every node
- Terminology:
  - Infected node: node with an update it is willing to spread
  - Susceptible node: node that is not yet updated

Propagation Models

- **Anti-entropy:** each node regularly chooses another node at random, and exchanges updates, leading to identical states at both afterwards
- **Gossiping:** A node which has just been updated (i.e., infected), tells a number of other nodes about its update (infecting them as well)