DISTRIBUTED SYSTEMS
Principles and Paradigms
Second Edition
ANDREW S. TANENBAUM
MAARTEN VAN STEEN

Chapter 4
Communication

*** Modified by Prof. Rivalino Matias, Jr.**

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Outline

- Network layered protocols (summary)
- IPC/RPC mechanisms
- Message-oriented communication
- Multicast communication

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Layered Protocols (1)

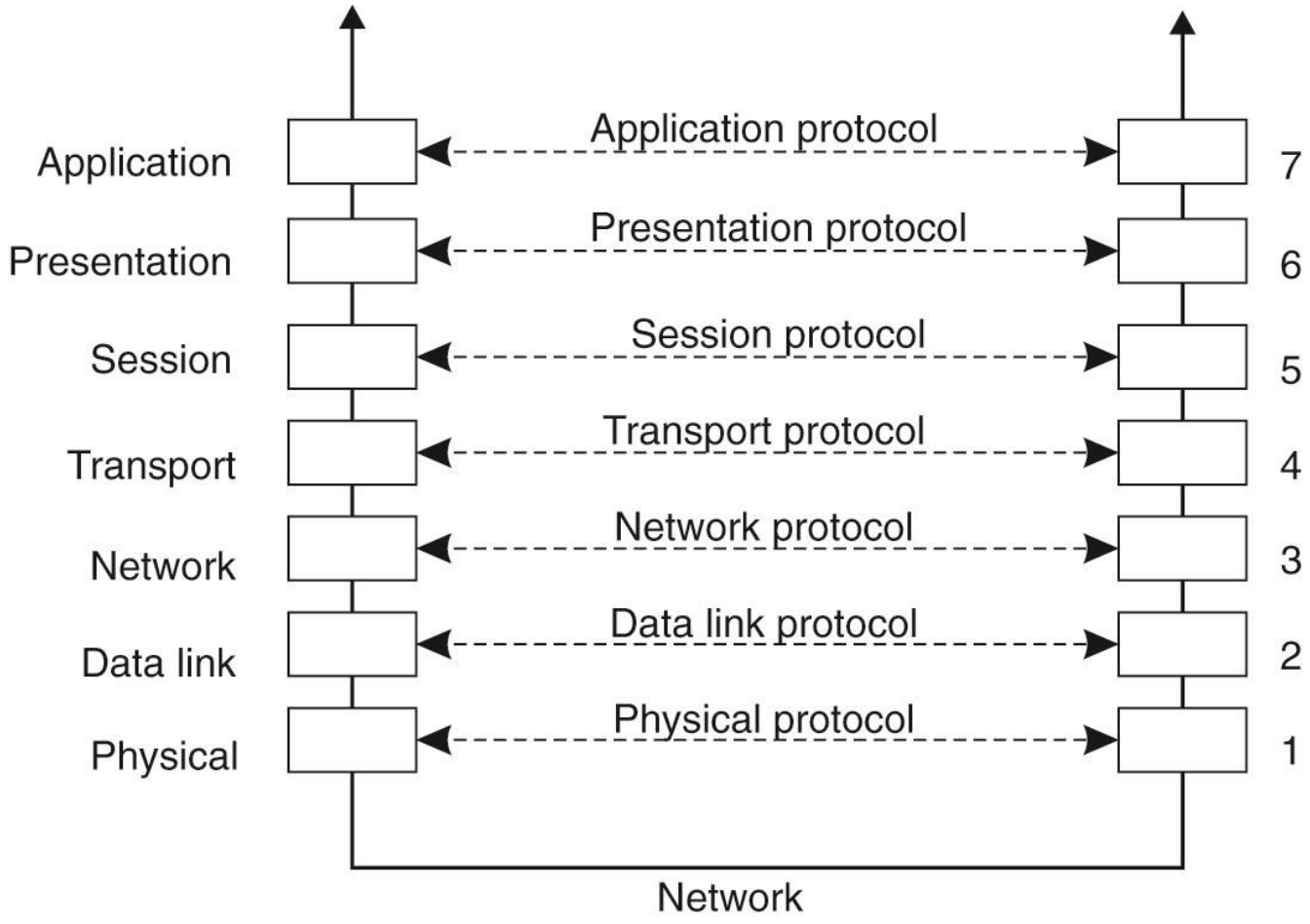


Figure 4-1. Layers, interfaces, and protocols in the OSI model.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Layered Protocols (2)

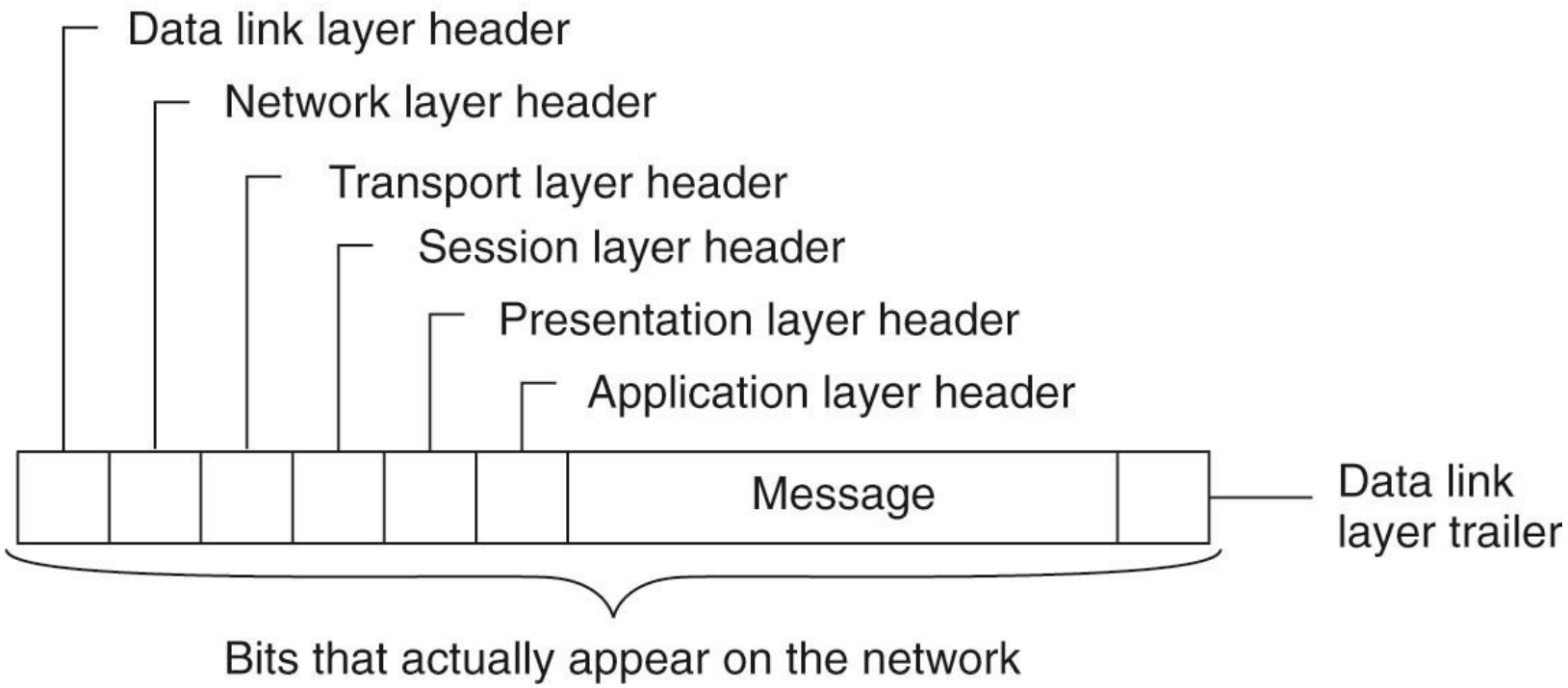


Figure 4-2. A typical message as it appears on the network.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Middleware Layer

- Middleware is invested to provide common services and protocols that can be used by many different applications. For example:
  - A rich set of communication protocols.
  - (Un)marshaling of data, necessary for integrated systems.
  - Naming protocols, to allow easy sharing of resources.
  - Scaling mechanisms, such as for replication and caching.
- Middleware's services are those mostly found in layers 5, 6 and 7 of the OSI model.
  - Remote procedure calls (L7)
  - (Un)marshaling of data (L6)
  - Naming protocols (L7)
  - Synchronization (L5)
  - …

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Middleware Protocols

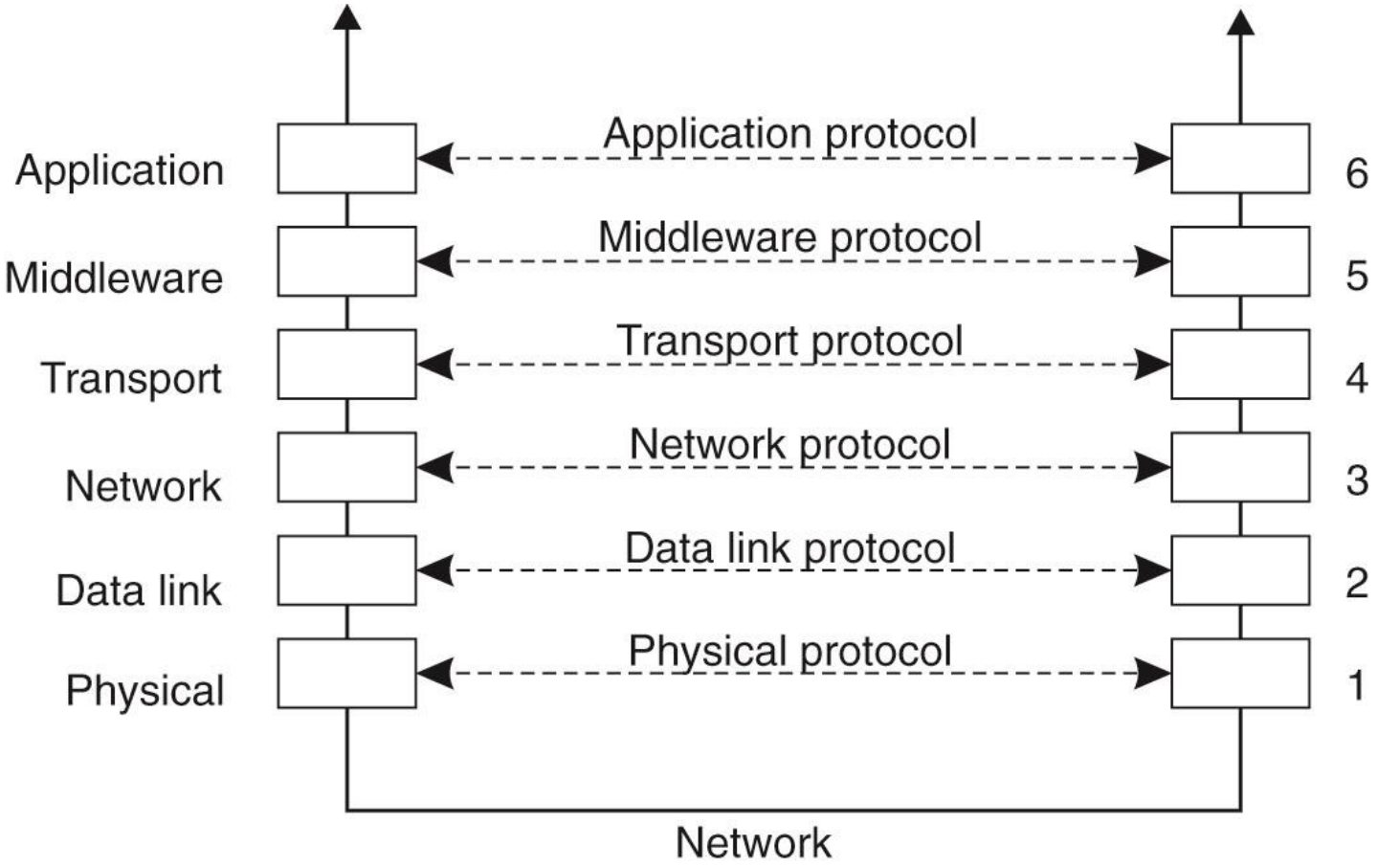


Figure 4-3. An adapted reference model for networked communication.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Types of Communication (1)

- Transient
  - With **transient communication**, a message is stored by the communication system (middleware) only if the sending and receiving application are executing.
  - If the middleware cannot deliver a message due to a transmission interrupt, or because the recipient is currently not active, it will simply be discarded.
- Persistent
  - With **persistent communication**, a message that has been submitted for transmission is stored by the communication middleware as long as it takes to deliver it to the receiver.
  - In this case, the middleware will store the message at one or several of the storage facilities as shown in Fig. 4.4 (next slide).

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Types of Communication (2)

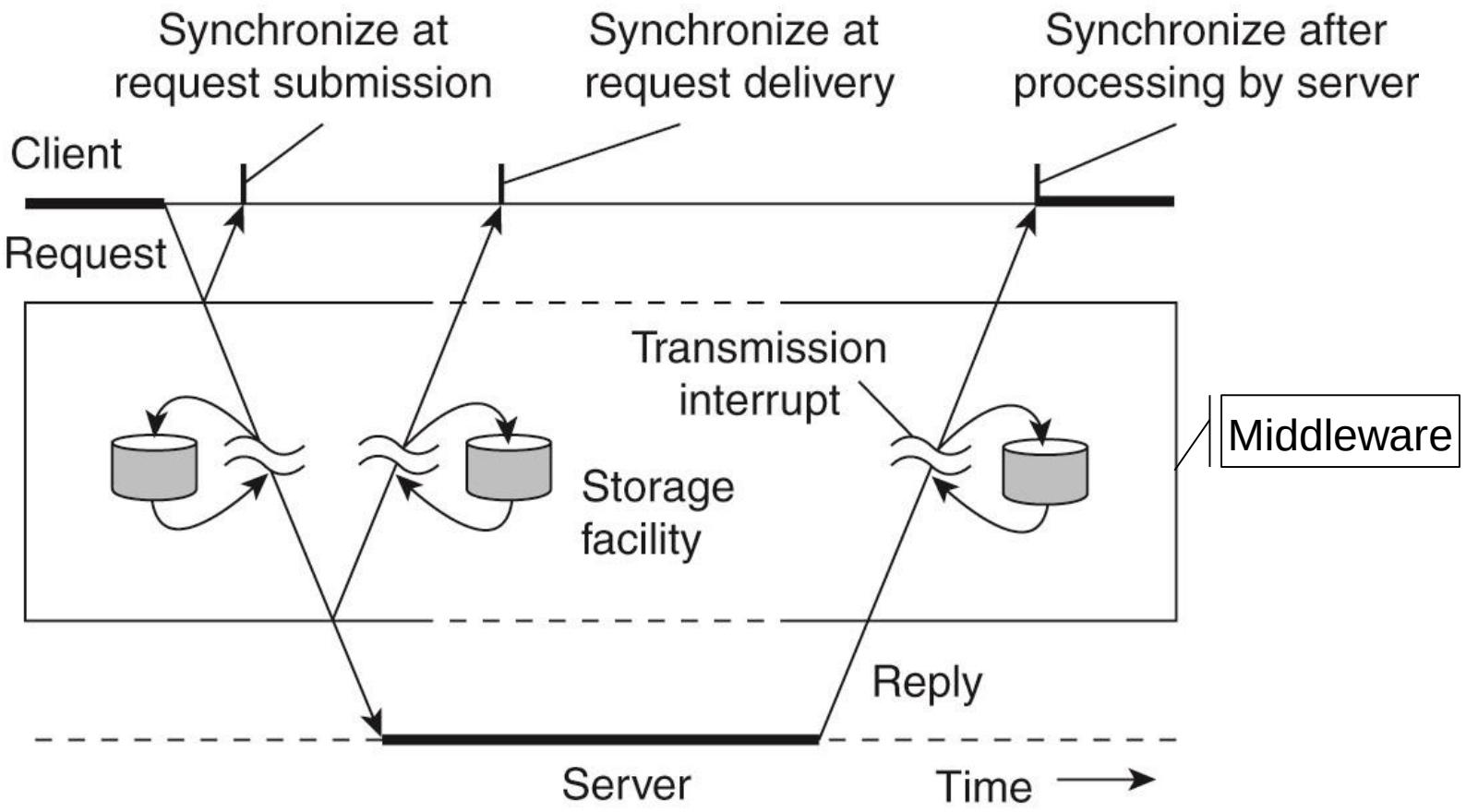


Figure 4-4. Viewing middleware as an intermediate (distributed) service in application-level communication.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Types of Communication (3)

- Synchronous vs. Asynchronous
  - With **synchronous communication**, the sender is blocked until its request is known to be accepted.
  - There are essentially three points where synchronization can take place:
    1) The sender may be blocked until the middleware notifies that it will take over transmission of the request.
    2) The sender may synchronize until its request has been delivered to the intended recipient.
    3) Synchronization may take place by letting the sender wait until its request has been fully processed, that is, up to the time that the recipient returns a response.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Types of Communication (4)

- Synchronous vs. Asynchronous
  - The characteristic feature of **asynchronous communication** is that a sender continues immediately after it has submitted its message for transmission.
  - This means that the message is (temporarily) stored immediately by the middleware upon submission.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Types of Communication (5)

- Combinations of persistence and synchronization occur in practice. Popular ones are:
  - **Persistence in combination with synchronization** at request submission, which is a common scheme for many message-queuing systems.
  - Likewise, **transient communication with synchronization** after the request has been fully processed is also widely used. This scheme corresponds with remote procedure calls (RPC).

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Types of Communication (6)

- **Client/Server** computing is generally based on a model of **transient synchronous communication**:
  - Client and server have to be active at time of communication.
  - Client issues request and blocks until it receives reply.
  - Server essentially waits only for incoming requests, and subsequently processes them.
- Drawbacks synchronous communication:
  - Client cannot do any other work while waiting for reply.
  - Failures must be handled immediately: *the client is waiting*…
  - The model may simply not be appropriate (mail, news)

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Types of Communication (7)

- **Message-oriented middleware**:
  - Aims at high-level **persistent asynchronous communication**:
    - Processes send each other messages, which are queued.
    - Sender need not wait for immediate reply but can do other things.
    - Middleware often ensures fault tolerance.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Remote Procedure Call (1)

- **Client-Server model** is a very natural way on how to engineer big (distributed) software systems.
  - Communication between remote peers, **Caller** and **Callee**, can be hidden by using proper procedure-call mechanism.
  - It helps to make more transparent the distribution aspects of the client-server programming.
- The idea behind RPC is to make a remote procedure call look as much as possible like a local call.
  - Application developers are familiar with simple procedure model.
  - Well-engineered procedures operate in isolation (black box).
  - There is no fundamental reason not to execute procedures on separate machine.
- Local Procedure Call vs. Remote Procedure Call
  - Parameter passing (value, reference), data type representation, …

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Conventional Procedure Call

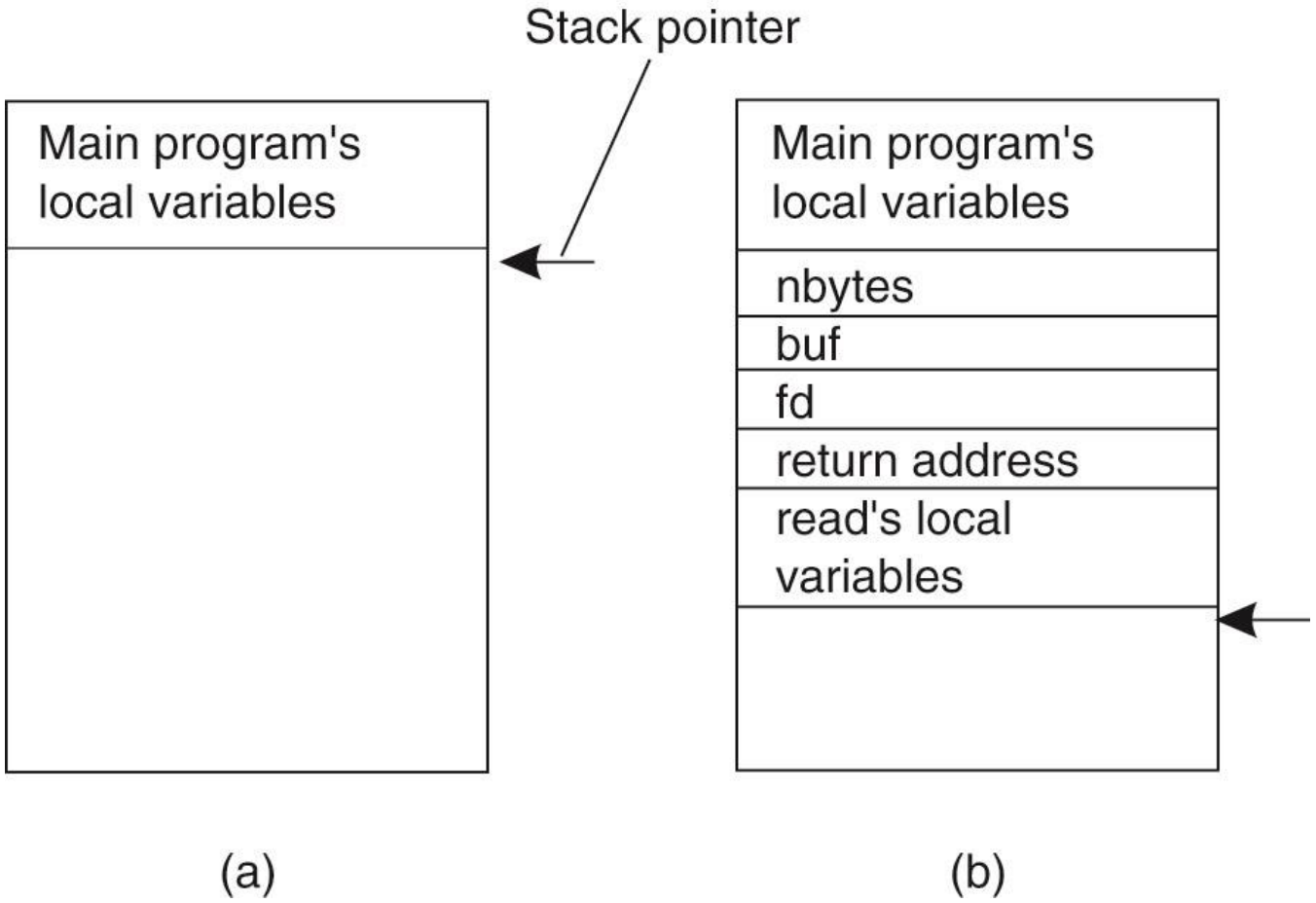


Figure 4-5. (a) Parameter passing in a local procedure call: the stack before the call to **read**. (b) The stack while the called procedure is active.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Client and Server Stubs

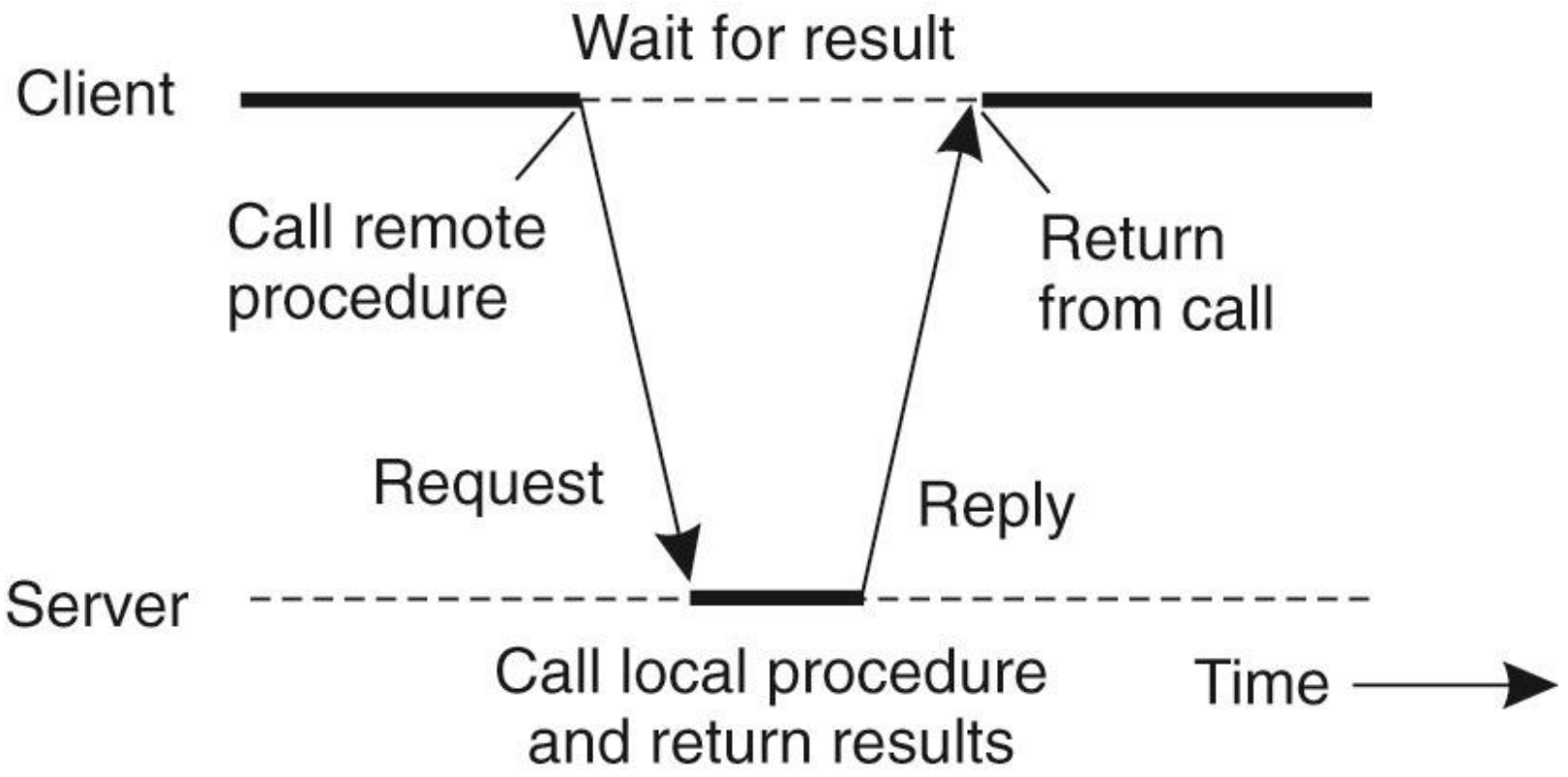


Figure 4-6. Principle of RPC between a client and server program.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Remote Procedure Calls (2)

A remote procedure call occurs in the following steps:

1. The client procedure calls the client stub in the normal way.
2. The client stub builds a message and calls the local operating system.
3. The client's OS sends the message to the remote OS.
4. The remote OS gives the message to the server stub.
5. The server stub unpacks the parameters and calls the server.

Continued …

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Remote Procedure Calls (3)

A remote procedure call occurs in the following steps (continued):

6. The server does the work and returns the result to the stub.
7. The server stub packs it in a message and calls its local OS.
8. The server's OS sends the message to the client's OS.
9. The client's OS gives the message to the client stub.
10. The stub unpacks the result and returns to the client.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Passing Value Parameters (1)

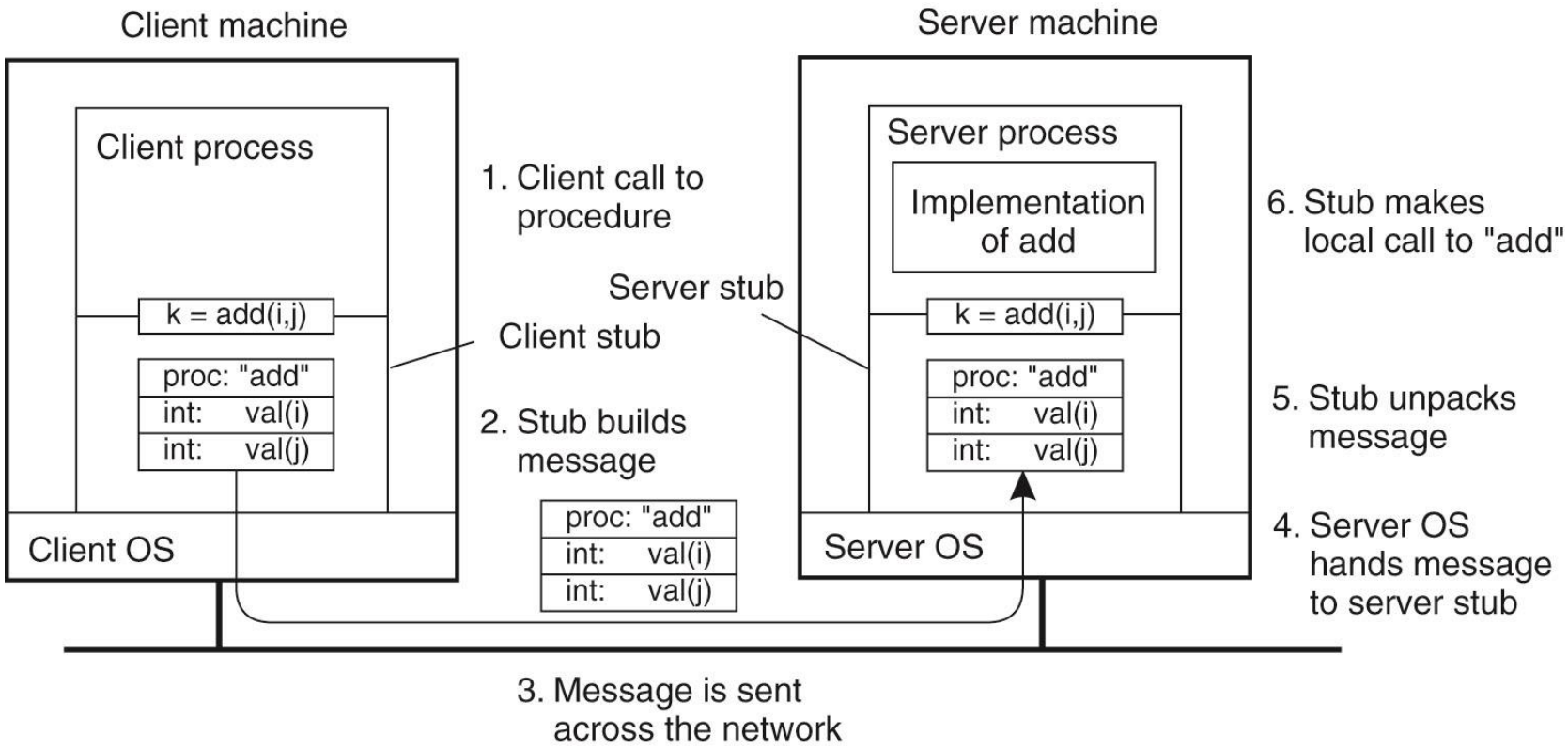


Figure 4-7. The steps involved in a doing a remote computation through RPC.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Passing Value Parameters (2)

- There's more than just wrapping parameters into a message:
  - Client and server machines may have different data representations (e.g., byte ordering).
  - Wrapping a parameter means transforming a value into a sequence of bytes (serialization).
  - Client and server have to agree on the same encoding:
    - How are basic data values represented (integers, floats, characters)
    - How are complex data values represented (arrays, structs, unions)
    - For this purpose, standards are necessaries:
      - ✓ E.g., ASN.1 (OSI L6), XDR (TCP/IP), XML, JSON, …
  - Client and server need to properly interpret messages, transforming them into machine-dependent representations.

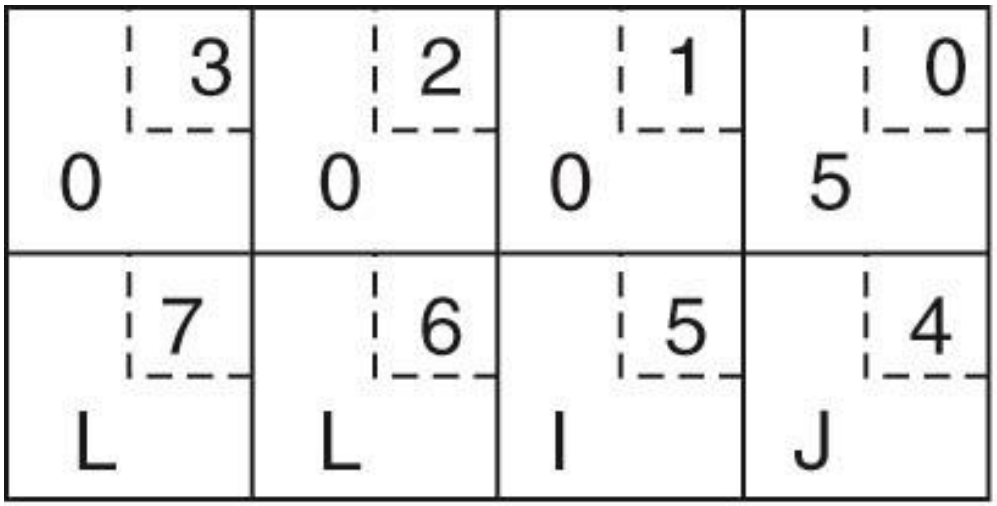
Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Passing Value Parameters (3)

- Example of problem with data representation (endianness):
  - *Endianness* is the order of bytes of a word in computer memory.
  - Intel/AMD adopt **little endian** (LE).
  - SPARC/Motorola processors adopt **big endian** (BE).
  - Consider an integer parameter (32-bit word) and a 4-byte string:
    - $5_d \rightarrow$ 0005 (**little endian**) or 5000 (**big endian**)
    - 'JILL' → 'LLIJ' (**little endian**) or 'JILL' (**big endian**) (stored in a word)

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Passing Value Parameters (4)

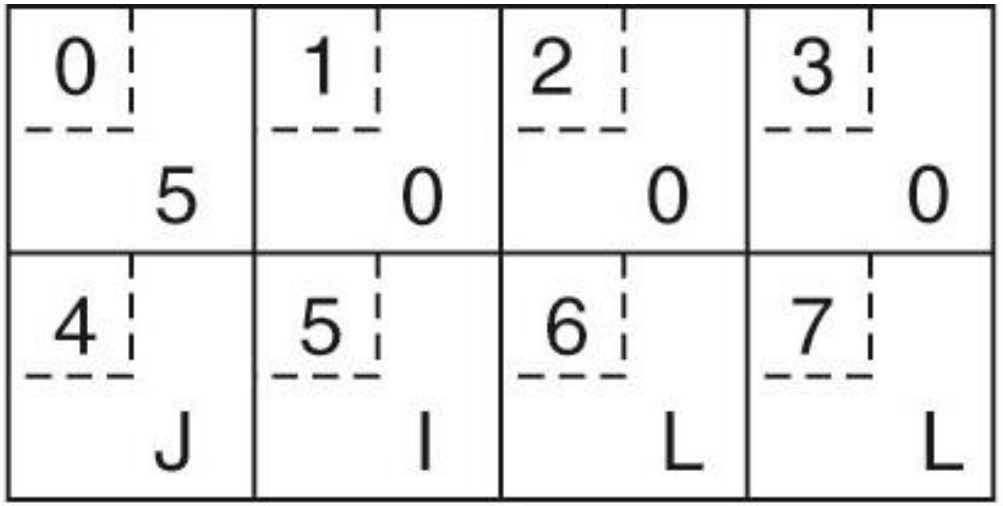


(a)

Figure 4-8. (a) The original message on the Pentium (LE).

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Passing Value Parameters (5)

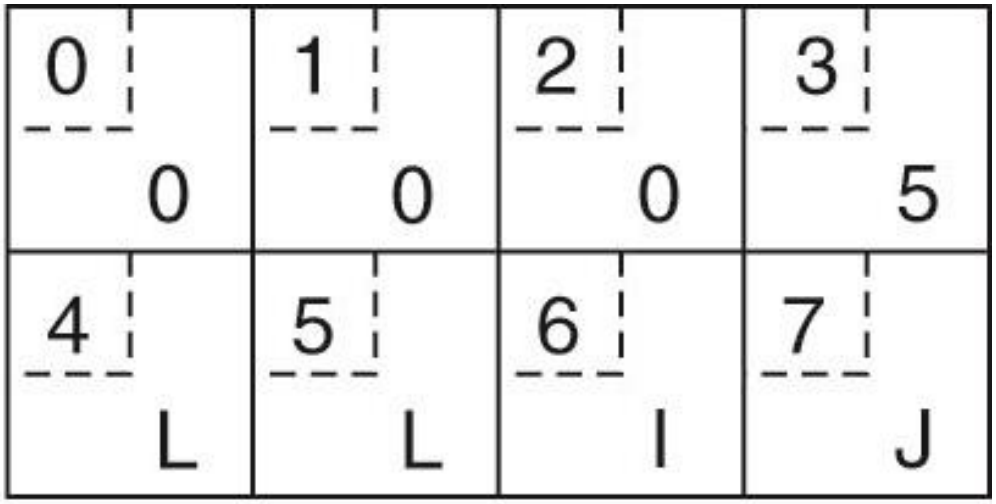


(b)

Figure 4-8. (a) Message on the SPARC (BE).

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Passing Value Parameters (6)



(c)

Figure 4-8. (c) The message after being inverted. The little numbers in boxes indicate the address of each byte.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Passing Value Parameters (7)

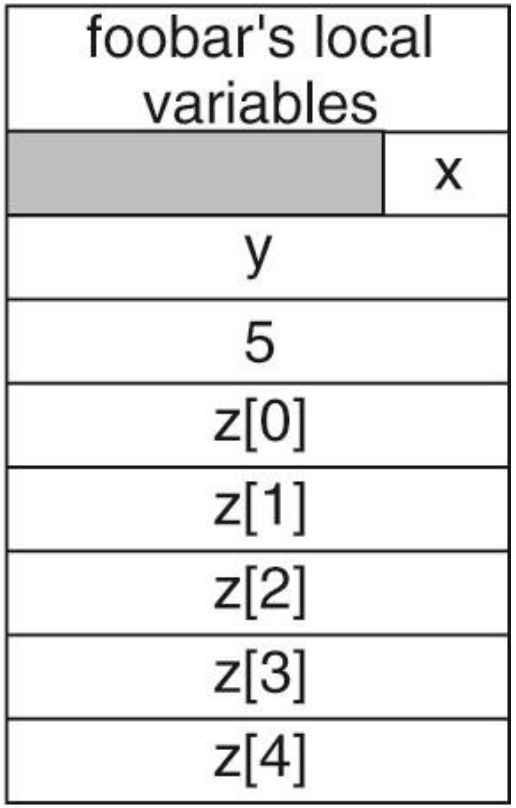
- Some assumptions:
  - Copy in/copy out semantics: while procedure is executed, nothing can be assumed about parameter values.
  - All data that is to be operated on is passed by parameters. Excludes passing references to (global) data.
- Conclusion:
  - Full access transparency cannot be realized.
  - A remote reference mechanism enhances access transparency:
    - Remote reference offers unified access to remote data.
    - Remote references can be passed as parameter in RPCs.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Parameter Specification and Stub Generation

```
foobar( char x; float y; int z[5] )
{
   ....
}
```

(a)



(b)

Figure 4-9. (a) A procedure. (b) The corresponding message.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Asynchronous RPC (1)

- Try to get rid of the strict request-reply behavior, but let the client continue without waiting for an answer from the server.

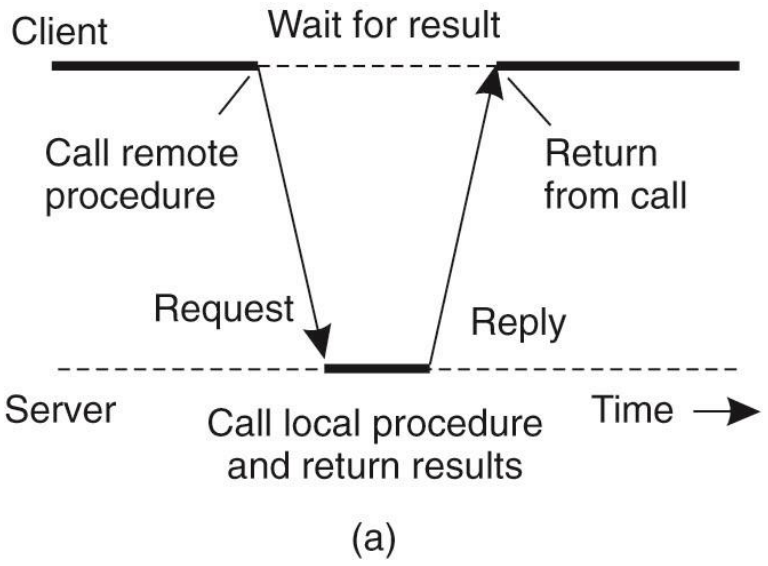


Figure 4-10. (a) The interaction between client and server in a traditional RPC.

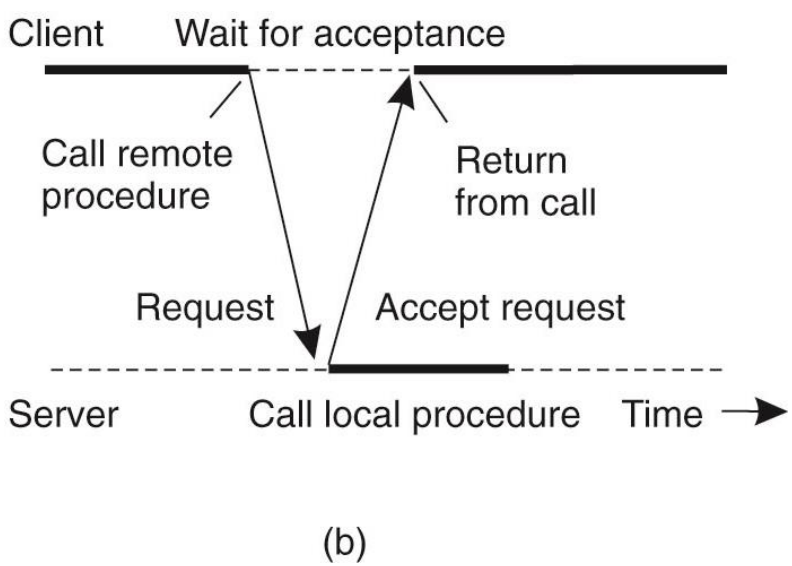


Figure 4-10. (b) The interaction using asynchronous RPC.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Asynchronous RPC (2)

- Try to get rid of the strict request-reply behavior, but let the client continue without waiting for an answer from the server (cont'd).

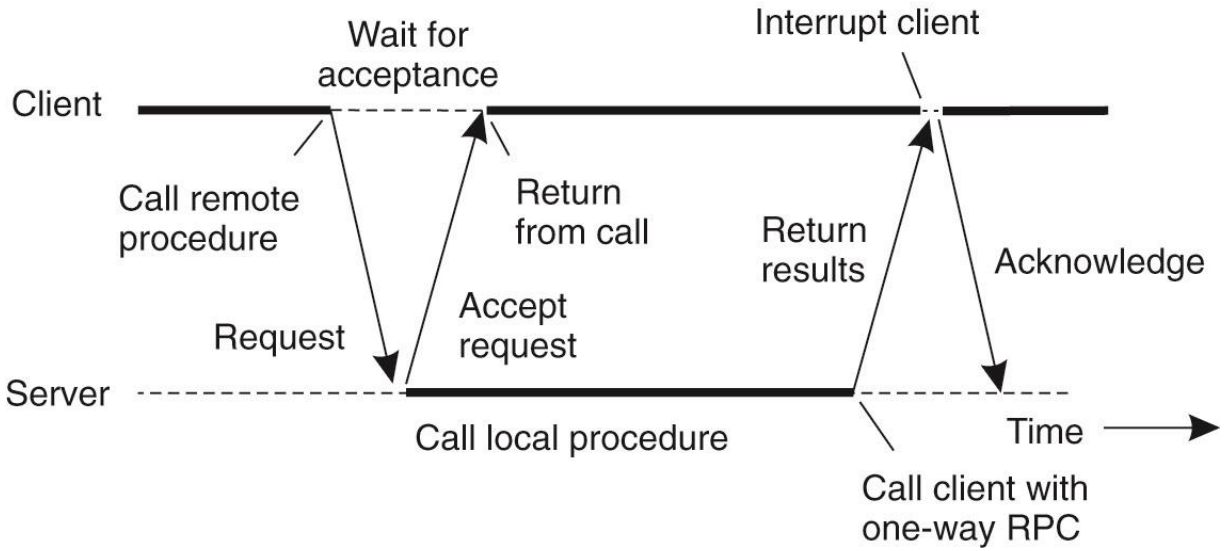


Figure 4-11. A client and server interacting through two asynchronous RPCs (deferred synchronous RPC).

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Writing a Client and a Server (1)

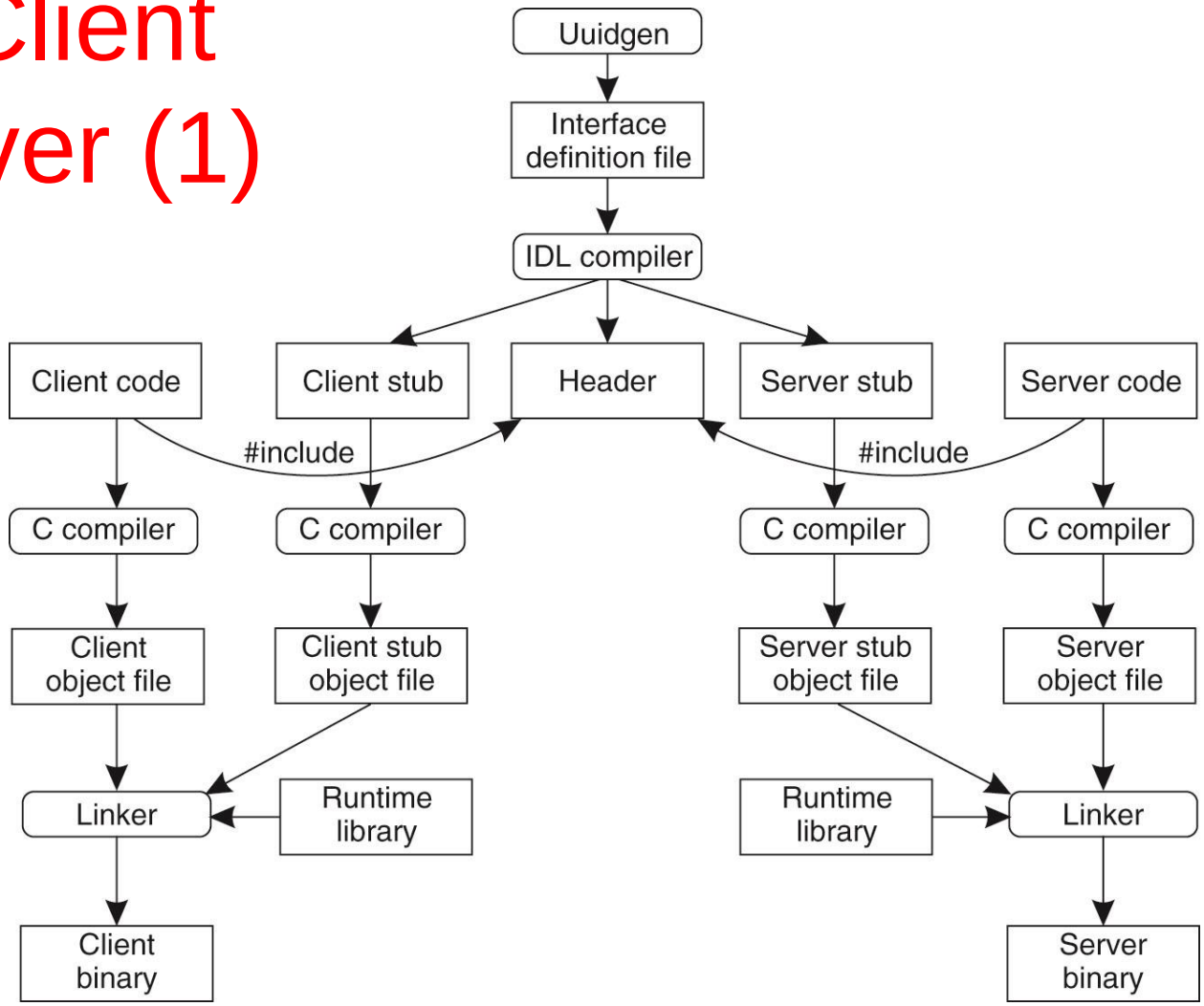


Figure 4-12. The steps in writing a client and a server in DCE RPC.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Writing a Client and a Server (2)

Three files output by the IDL compiler:

- A header file (e.g., interface.h, in C terms).
- The client stub.
- The server stub.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Binding a Client to a Server (1)

- Registration of a server makes it possible for a client to locate the server and bind to it.

- Server location is done in two steps:
  1. Locate the server's machine.
  2. Locate the server on that machine.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Binding a Client to a Server (2)

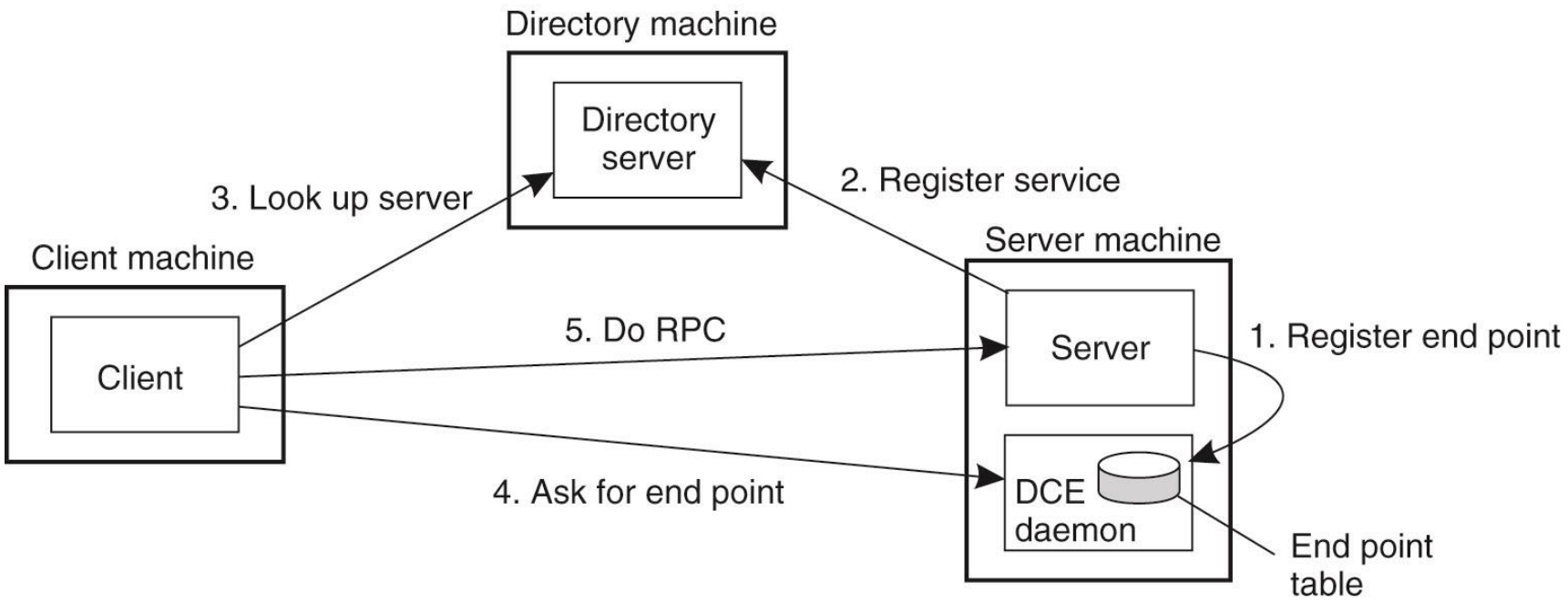


Figure 4-13. Client-to-server binding in DCE.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Message-oriented Communication

- **Transient messaging**
  - Sockets
  - MPI
- **Persistent messaging**
  - Message-queuing (MQ) systems
  - Brokers

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Berkeley Sockets (1)

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

Figure 4-14. The socket primitives for TCP/IP.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Berkeley Sockets (2)

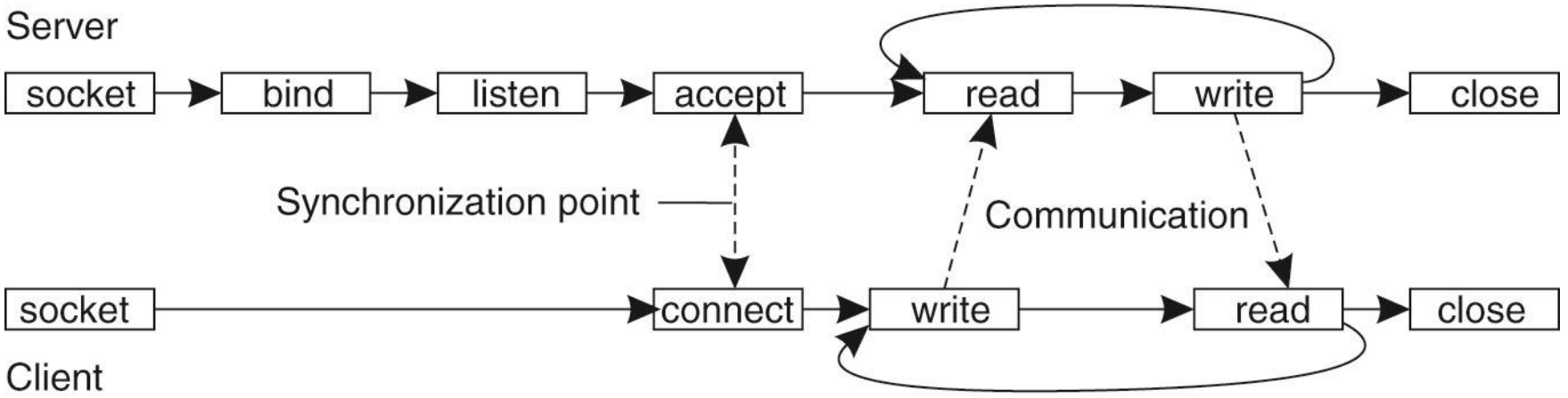


Figure 4-15. Connection-oriented communication pattern using sockets.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

The Message-Passing Interface

Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and wait until receipt starts
MPI_recv	Receive a message; block if there is none
MPI_irecv	Check if there is an incoming message, but do not block

Figure 4-16. Some of the most intuitive message-passing primitives of MPI.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Persistent messaging

- Can be achieved through support of middleware-level queues.
  - **Queues** correspond to buffers at communication servers.
  - **Asynchronous persistent** communication is supported.
  - Some models allow sender and receiver **be decoupled in time and space**.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Message-Queuing Model (1)

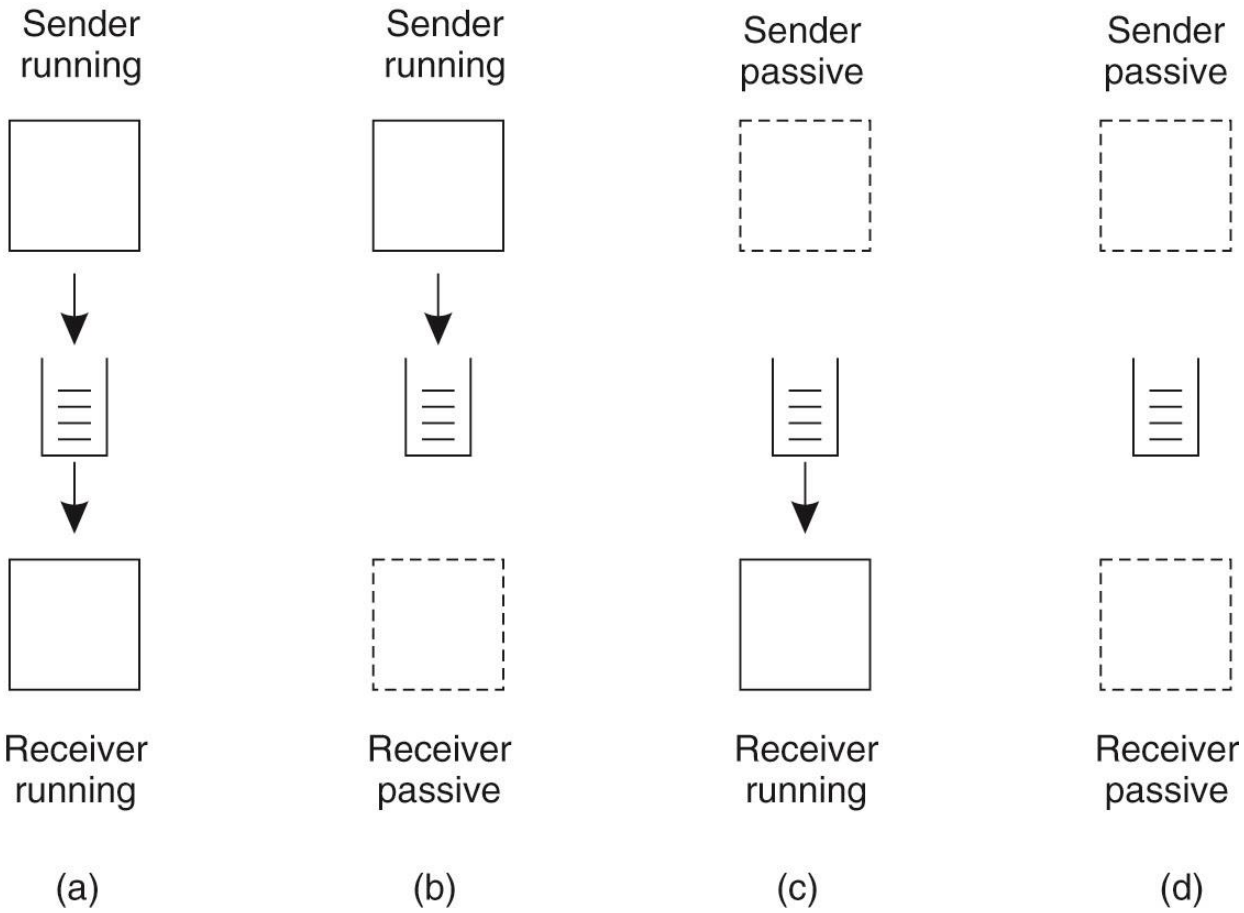


Figure 4-17. Four combinations for loosely-coupled communications using queues.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Message-Queuing Model (2)

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

Figure 4-18. Basic interface to a queue in a message-queuing system.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

General Architecture of a Message-Queuing System (1)

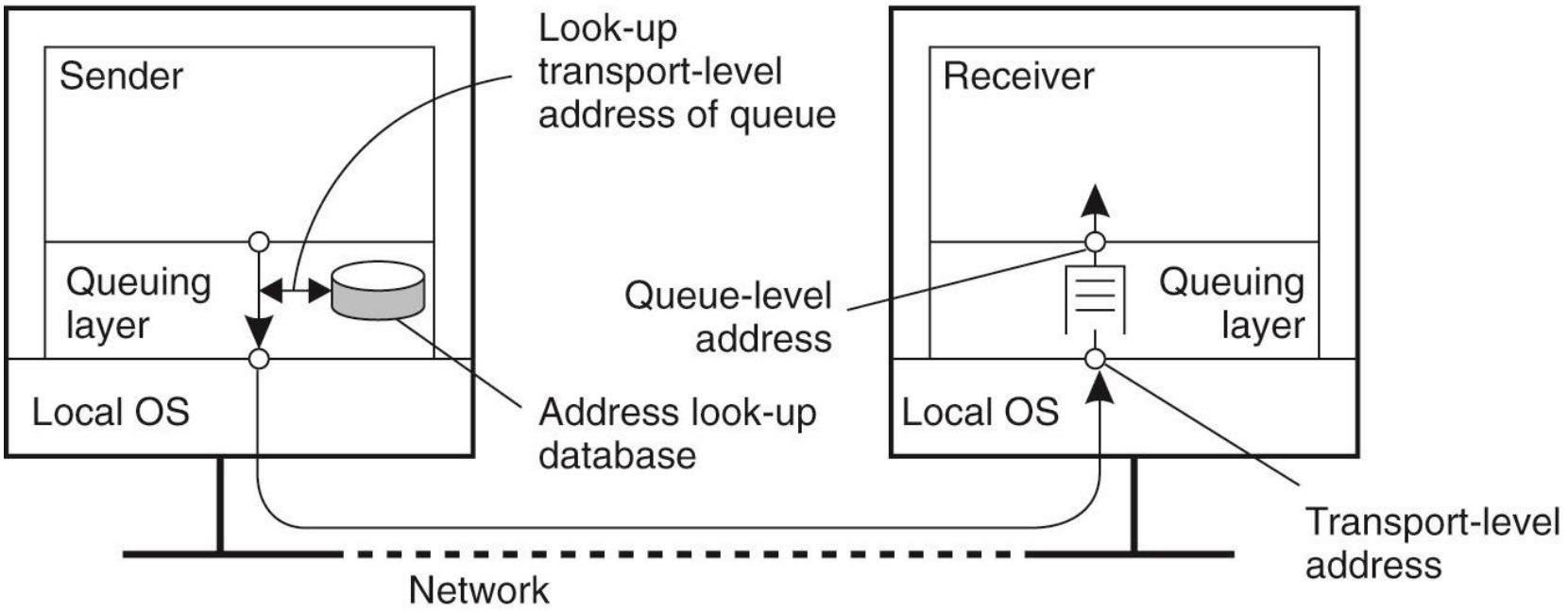


Figure 4-19. The relationship between queue-level addressing and network-level addressing.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

General Architecture of a Message-Queuing System (2)

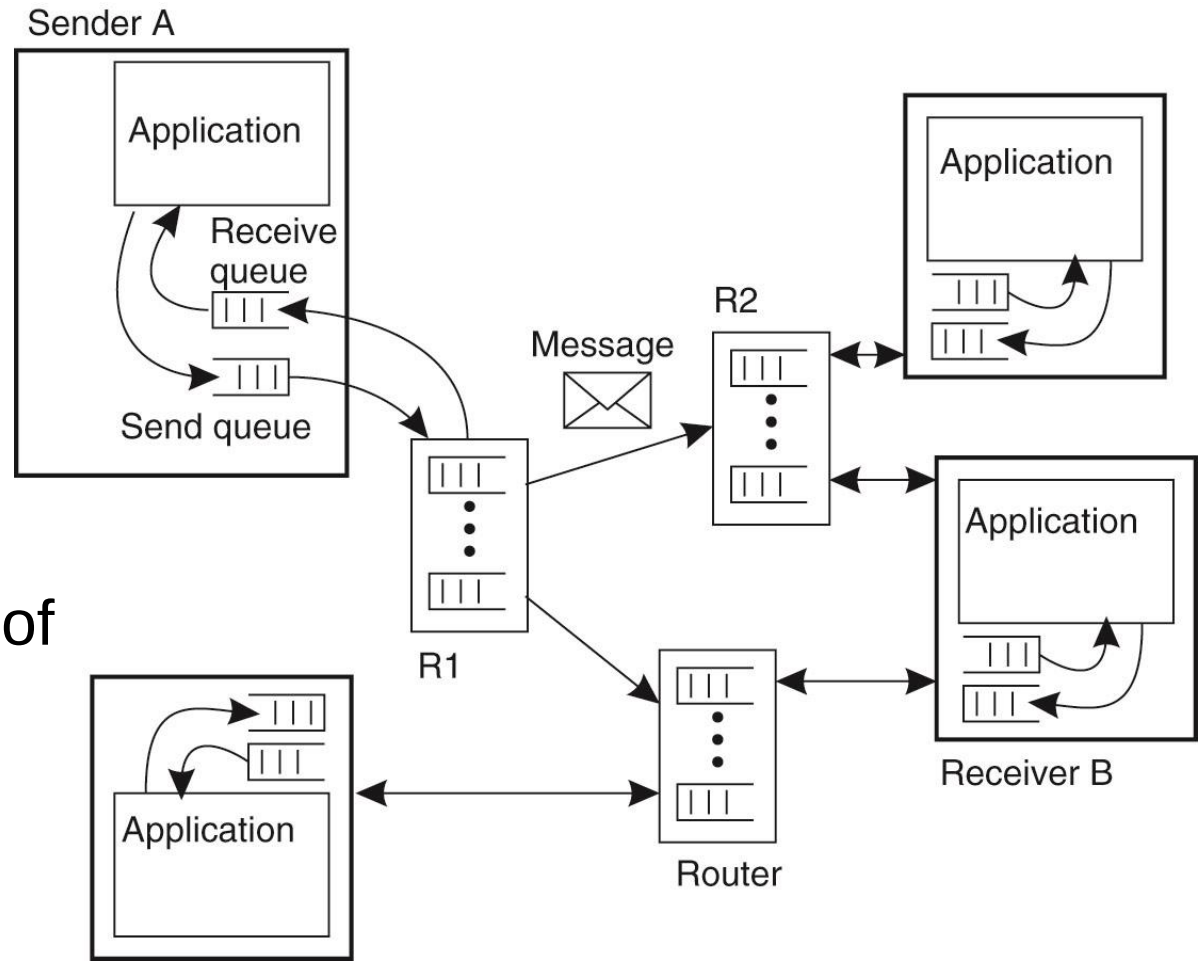


Figure 4-20. The general organization of a message-queuing system with routers.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Message Brokers

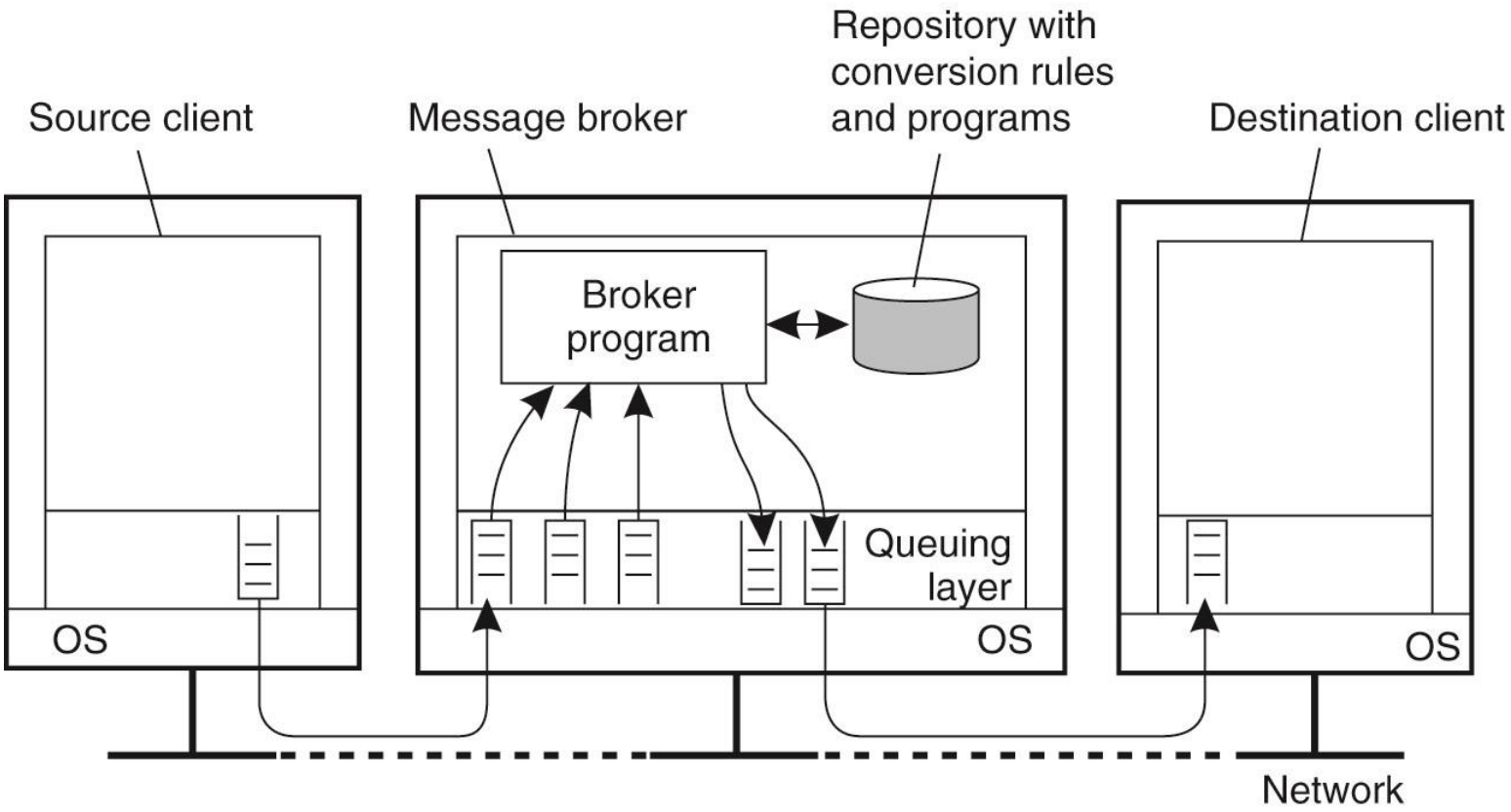


Figure 4-21. The general organization of a message broker in a message-queuing system.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

IBM's WebSphere Message-Queuing System

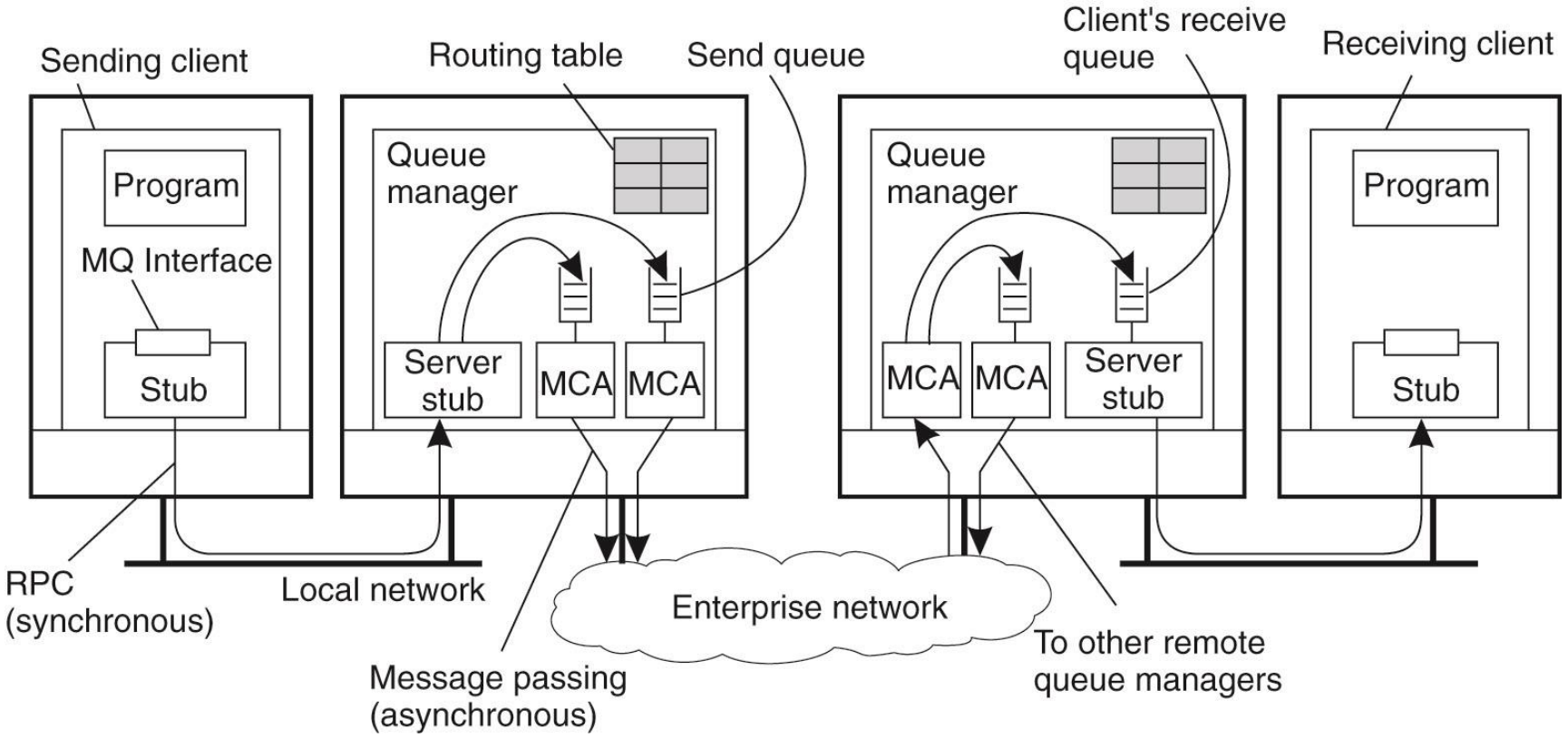


Figure 4-22. General organization of IBM's message-queuing system.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Channels

Attribute	Description
Transport type	Determines the transport protocol to be used
FIFO delivery	Indicates that messages are to be delivered in the order they are sent
Message length	Maximum length of a single message
Setup retry count	Specifies maximum number of retries to start up the remote MCA
Delivery retries	Maximum times MCA will try to put received message into queue

Figure 4-23. Some attributes associated with message channel agents.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Message Transfer (1)

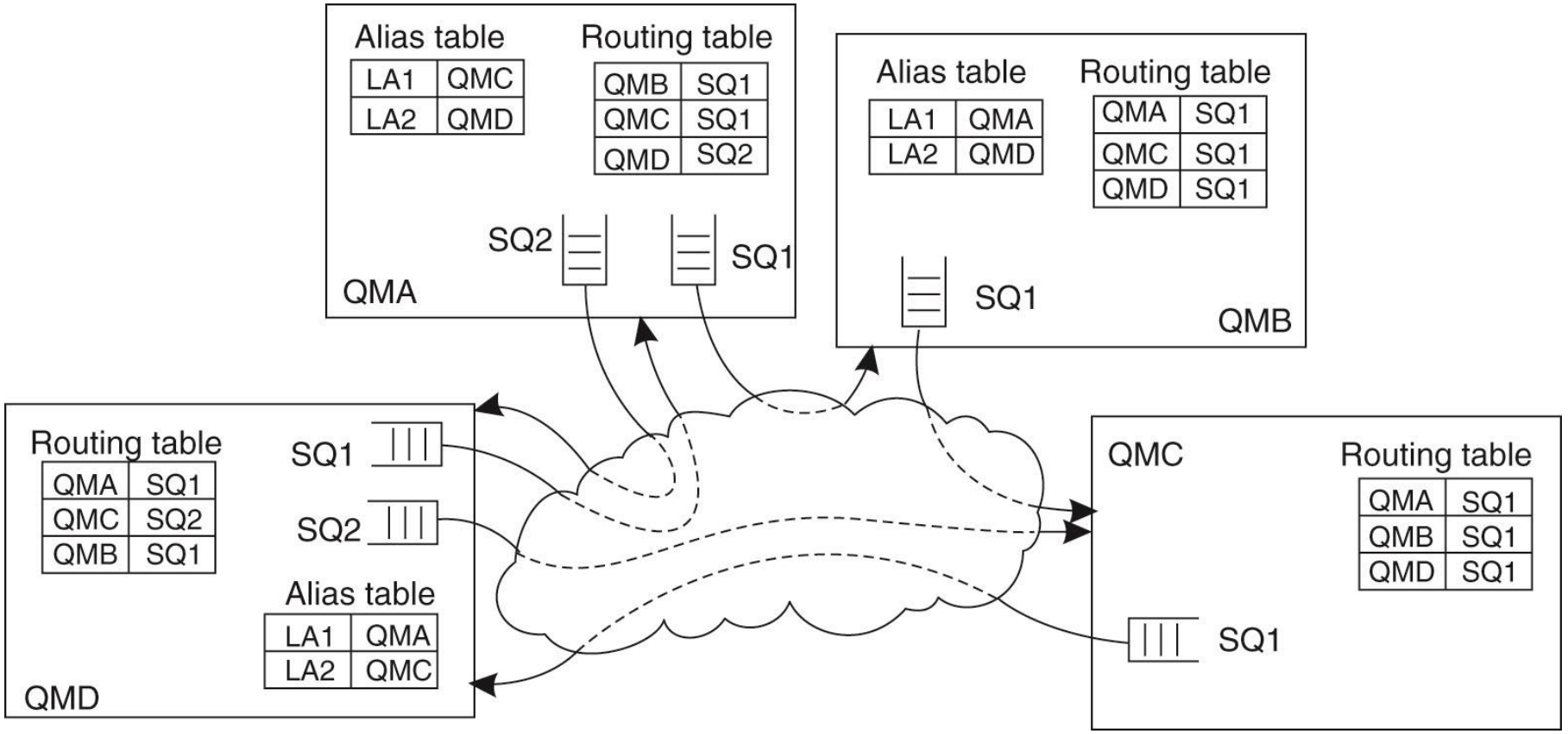


Figure 4-24. The general organization of an MQ queuing network using routing tables and aliases.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Message Transfer (2)

Primitive	Description
MQopen	Open a (possibly remote) queue
MQclose	Close a queue
MQput	Put a message into an opened queue
MQget	Get a message from a (local) queue

Figure 4-25. Primitives available in the message-queuing interface.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Multicast Communication

- Multicast
  - Application-level multicast
  - Performance issues in overlay
  - Flooding-based multicast
  - Gossiping-based dissemination

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Multicast

- Application-level Multicast
  - For many years, studies in multicasting has belonged to the domain of network protocols, especially for L3 and L4.
  - A major issues in all this effort was setting up the communication paths information dissemination.
    - In practice, this involved a huge management effort, requiring human intervention.
  - With the advance of P2P technology, and notably structured overlay networks, it became easier to set up communication paths.
  - As peer-to-peer solutions are typically developed at the application layer, various application-level multicast techniques have been introduced.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Multicast

- Application-level Tree-based Multicast
  - The basic idea is that nodes are organized into an overlay network, which is then used to disseminate information to its members.
  - An important observation is that network routers are not involved in group membership.
    - As a consequence, the connections between nodes in the overlay network may cross several physical links, which may not be optimal.
  - Two approaches to construct the overlay network:
    - Nodes organize themselves directly into a tree, meaning that there is a unique overlay path between every pair of nodes (**< latency**).
    - Nodes organize into a mesh networks, where in general there exist multiple paths between every pair of nodes (**> robustness**).

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Multicast

- Performance issues in overlays
  - Building a tree by itself is not that difficult once we have organized the nodes into an overlay.
    - Building an efficient tree may be a different story.
    - It should consider performance metrics.
  - Many solution are based purely on the logical routing of messages through the overlay.
    - Not considering the physical paths the message will pass through.
  - The quality of an application-based multicast tree is generally measured by three different metrics:
    - **Link stress:** is defined per link and counts how often a packet crosses the same link. A link stress > 1 comes from the fact that although at a logical level a packet may be forwarded along two different connections, part of those connections may correspond to the same physical link.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Overlay Construction

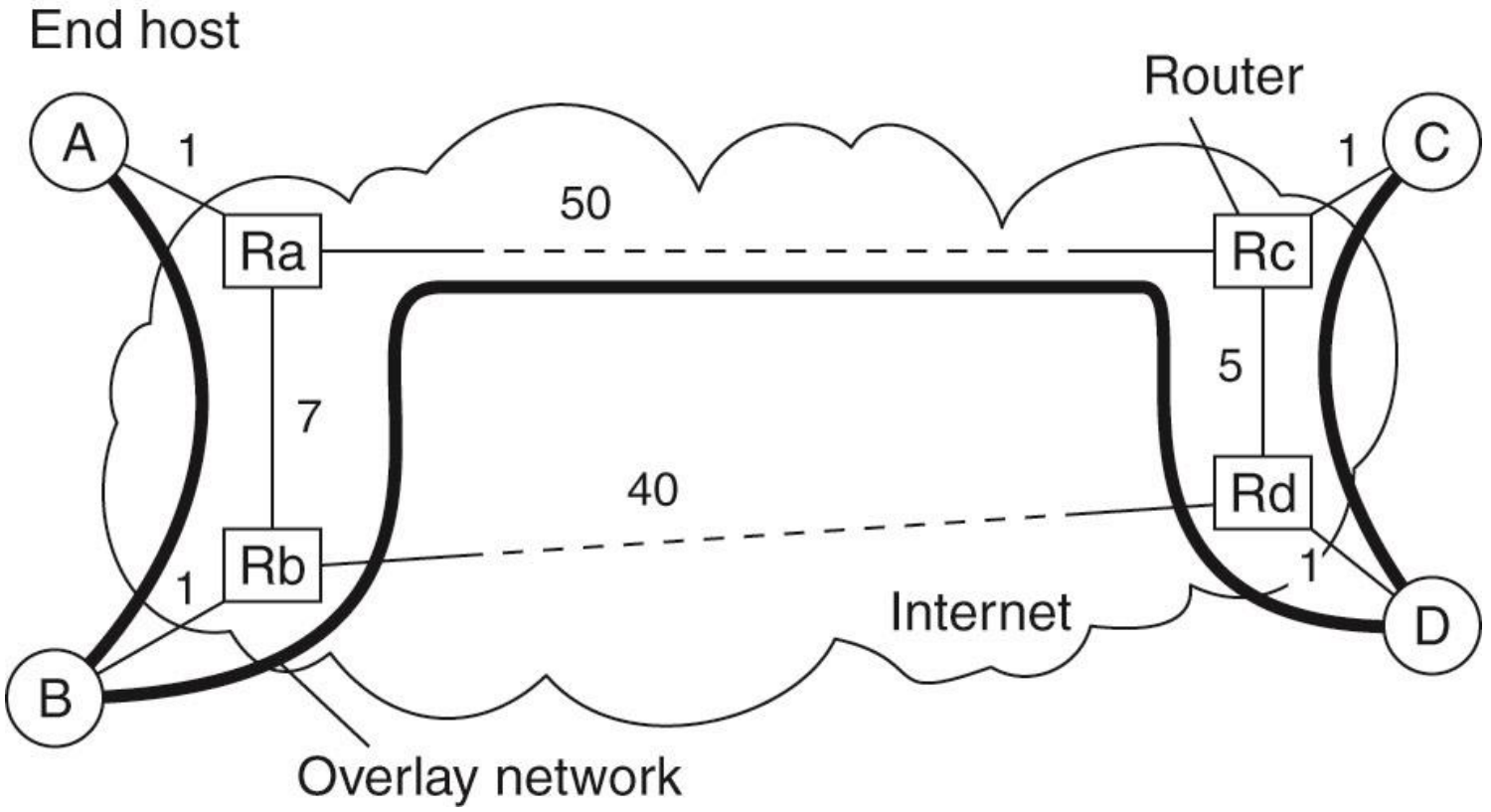


Figure 4-31. The relation between links in an overlay and actual network-level routes.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Multicast

- Performance issues in overlays
  - The quality of an application-based multicast tree is generally measured by three different metrics: (cont'd)
    - **Link stress**
    - **Stretch** or **relative delay penalty** (**RDP**): measures the ratio in the delay between two nodes in the overlay, and the delay that those two nodes would expect in the underlying network.

      For example, messages from B to C follow the route:

      B→Rb→Ra→Re→E→Re→Rc→Rd→D→Rd→Rc→C in the overlay network, having a total cost of 73 units.

      However, messages would have been routed in the underlying network along the path **B→Rb→Rd→Rc→C**, with a total cost of 47 units, leading to a **stretch** of 1.55. Obviously, when constructing an overlay network, the goal is to minimize the aggregated stretch.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Multicast

- Performance issues in overlays
  - The quality of an application-based multicast tree is generally measured by three different metrics: (cont'd)
    - **Link stress**
    - **Stretch** or **relative delay penalty** (**RDP**)
    - **Tree cost:** is a global metric, generally related to minimizing the aggregated link costs. For example, if the cost of a link is taken to be the delay between its two end nodes, then optimizing the **tree cost** reduces to finding a minimal spanning tree in which the total time for disseminating information to all nodes is minimal.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Multicast

- Flooding-based multicast
  - So far, it has been assumed that when a message is to be multicast, it is to be received by every node in the overlay network.
    - Only the nodes on the destination group process the message.
    - This physically correspond to broadcasting.
  - A key design issue when it comes to multicast is to minimize the use of intermediate nodes to which the message is not intended.
  - A simple way to avoid such inefficiency, is to construct an overlay network per *multicast group*.
    - As a consequence, multicasting a message **m** to a group **G** is the same as broadcasting **m** to **G**.
    - This is the case of flooding-based multicast.
  - To understand the performance of flooding, consider an overlay network as a connected graph G with N nodes and M edges.
    - Note that flooding means that we need to send (at least) M messages. If G is a tree, flooding is optimal, $M = N - 1$.
    - When G is fully connected, $M = 1 \div 2 \cdot N \cdot (N-1)$ messages (worst-case).

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Multicast

- Gossiping-based multicast
  - It is based on **epidemic behavior** (aka **gossiping**).
  - The main goal of this type of protocol is to rapidly propagate information among a large collection of nodes, using only local information.
    - There is no central component to coordinate the dissemination.
  - Terminology:
    - A node that is part of a distributed system is called **infected** if it holds data that it is willing to spread to other nodes.
    - A node that has not yet seen this data is called **susceptible**.
    - An updated node that is not willing or able to spread its data is said to have been **removed**.
  - A popular propagation model is that of **anti-entropy**.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Information Dissemination Models (1)

- **Anti-entropy propagation model**
  - Node *P* picks another node *Q* at random
  - Subsequently exchanges updates with *Q*
  - Approaches to exchanging updates
    - P only pushes its own updates to Q
    - P only pulls in new updates from Q
    - P and Q send updates to each other

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Information Dissemination Models (2)

- When it comes to rapidly spreading updates, only pushing updates turns out to be a bad choice.
  - Updates can be propagated only by infected nodes.
  - If many nodes are infected, the probability to select a susceptible node is relatively small.
- The pull-based approach works better when many nodes are infected.
  - Spreading updates is essentially triggered by susceptible nodes.
  - Chances are big that such a node will contact an infected node.
- The push-pull revels the best strategy.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5