Distributed Systems

Communication in Distributed Systems

WEEK 6: Message-Oriented Communication, Remote Procedure Call (RPC), Sockets, Protocols,

Online Lecture Series - 6

Felix Edesa

Addis Ababa Science and Technology University

AASTU

Course Outline: Communication

Topics We Will Cover

- **Foundations** — Layered Protocols, Types of Communication
- **Remote Procedure Call (RPC)** — Basics, Parameter Passing, Application Support, Variations
- **Message-Oriented Communication** — Sockets, Advanced Messaging, Persistent Messaging, AMQP
- **Multicast Communication** — Tree-Based, Flooding, Gossip-Based Dissemination
- **Summary** — Key Takeaways

AASTU

Foundations of Communication

Key Idea

- In distributed systems, all communication is based on message passing.
- Processes must agree on the meaning of transmitted bits through protocols.

Practical Scenarios

Email, Video Call, Online Shopping: Examples of applications that rely on agreed communication protocols for correct and secure operation.

AASTU

Why Layered Protocols?

Motivation

- Without structure, communication becomes chaotic.
- Layers divide complex communication into manageable parts.
- Example: Watching Netflix involves multiple layers — video streaming (application), TCP reliability (transport), and IP routing (network).

AASTU

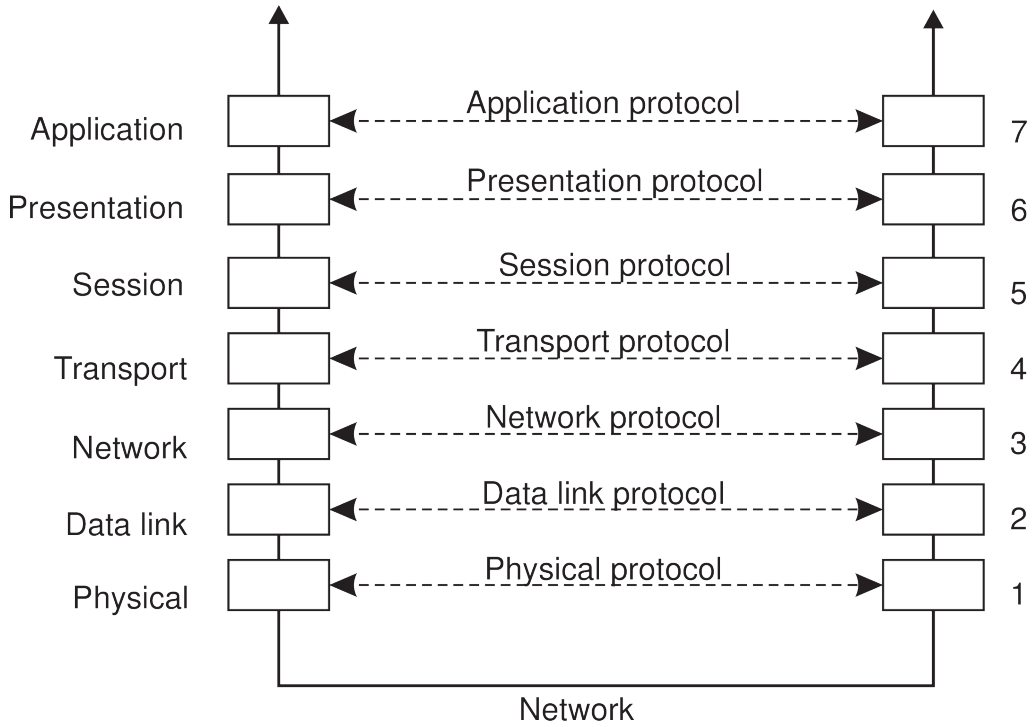
The OSI Reference Model

Background

- Developed by ISO (Day and Zimmerman, 1983).
- Provides 7 abstraction layers for communication.
- Example: Sending an email uses Application (SMTP), Transport (TCP), Network (IP), and lower layers.

AASTU

Basic networking model



Maarten van Steen and Andrew S. Tanenbaum, *Distributed Systems*, Fourth edition, Version 4.02, February 2024.

OSI Layers Overview

Key Idea

- Communication in distributed systems relies on layered protocols.
- Protocols define rules for sending/receiving messages.
- Example: TCP/IP stack in the Internet ensures reliable communication.

AASTU

Lower-Level Protocols: Physical and Data Link Layers

Physical Layer

- Transmits raw bits; defines voltage, bit rate, connector types.
- Example: USB, Ethernet cables.

Data Link Layer

- Groups bits into frames, detects errors with checksums.
- Example: Wi-Fi ensures correct reception of frames over wireless networks.

AASTU

Network Layer

Key Idea

- Routes messages between devices across multiple networks.
- Example: IP packets are routed independently to the destination.

Practical Scenario

- Internet: When you load a webpage, your request travels through multiple routers to the server.

AASTU

Transport Layer

Key Idea

- Provides reliable or connectionless delivery of messages to applications.
- Example: TCP ensures email or web requests arrive correctly and in order.

Other Protocols

- UDP for streaming video/calls; RTP for real-time streaming; SCTP for message-oriented delivery.

AASTU

Session Layer

Key Idea

- Controls dialog between applications; manages checkpoints.
- Keeps track of which side is currently communicating.

Practical Example

- Video conferencing: Zoom tracks active speakers and session state.

AASTU

Presentation Layer

Key Idea

- Handles data format translation and encoding.
- Ensures different systems understand each other's data.

Practical Example

- Email attachments: converting images to standard formats (JPEG/PNG) before sending.

AASTU

Application Layer

Key Idea

- Provides protocols for end-user applications.
- Supports email, file transfer, web services, remote access, and messaging apps.
- Enables interoperability across different devices and platforms.

Practical Examples

- HTTP for web browsing, FTP for file transfer, SMTP for sending emails.

AASTU

Middleware Protocols

Purpose

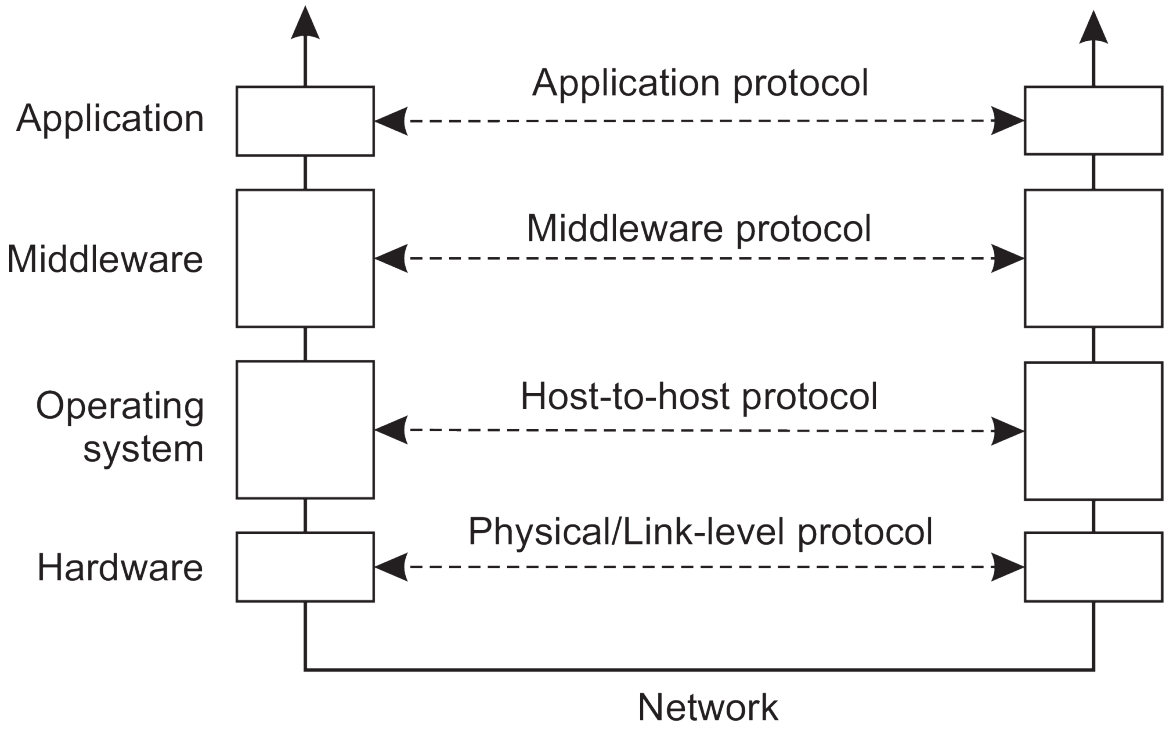
- Provides shared services and protocols to simplify communication

Key Features

- Rich set of communication protocols for distributed apps.
- (Un)marshaling of data for interoperability.
- Naming protocols for easy resource sharing.
- Security protocols to ensure safe communication.
- Scaling mechanisms: replication, caching.

AASTU

An adapted layering scheme



Maarten van Steen and Andrew S. Tanenbaum, *Distributed Systems*, Fourth edition, Version 4.02, February 2024.

Types of Communication

Key Categories

- **Persistent Communication:** Message stored until receiver is available (e.g., Email).
- **Transient Communication:** Message discarded if receiver not active (e.g., UDP).
- **Asynchronous Communication:** Sender continues after submitting message.
- **Synchronous Communication:** Sender waits until request is accepted, delivered, or processed.

AASTU

Practical Examples of Communication

Scenarios

- ⋆ **Email (Persistent):** Message is stored on mail server until user retrieves it.
- ⋆ **Chat Apps (Transient):** If offline, message may be lost unless stored by the app.
- ⋆ **Messaging Queue (Async):** Kafka/RabbitMQ store messages and process later.
- ⋆ **Video Call (Sync):** User must wait for live response.

AASTU

Remote Procedure Call (RPC)

Concept

- Allows a program to call a procedure on another machine as if it were local.
- Hides message passing from programmers $\rightarrow$ promotes transparency.
- Example: Client requests `append(data, dbList)` on a remote server.

AASTU

Why RPC is Important

Benefits

- Simplifies distributed programming by hiding network details.
- Makes remote operations look like local calls.
- Widely used in microservices, databases, and cloud systems.

Practical Example

- ⋆ **Database Access:** Client inserts a record remotely without writing network code.
- ⋆ **Cloud APIs:** Google Drive or AWS services use RPC-like mechanisms.

AASTU

Challenges in RPC

Key Issues

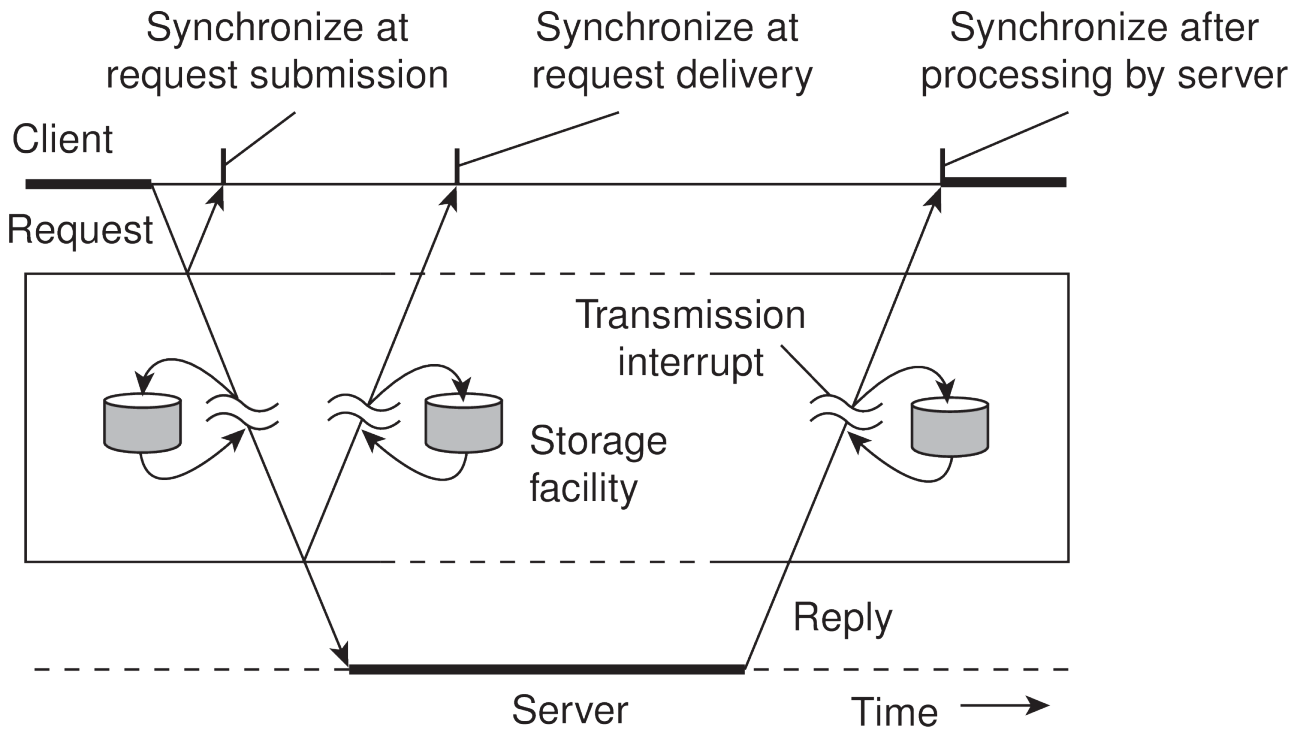
- Different address spaces (client vs. server).
- Data conversion (e.g., integer sizes on different machines).
- Failure handling (client crash, server crash, network errors).

Practical Note

- RPC failures can look like “frozen apps” or “timeouts” to the user.

AASTU

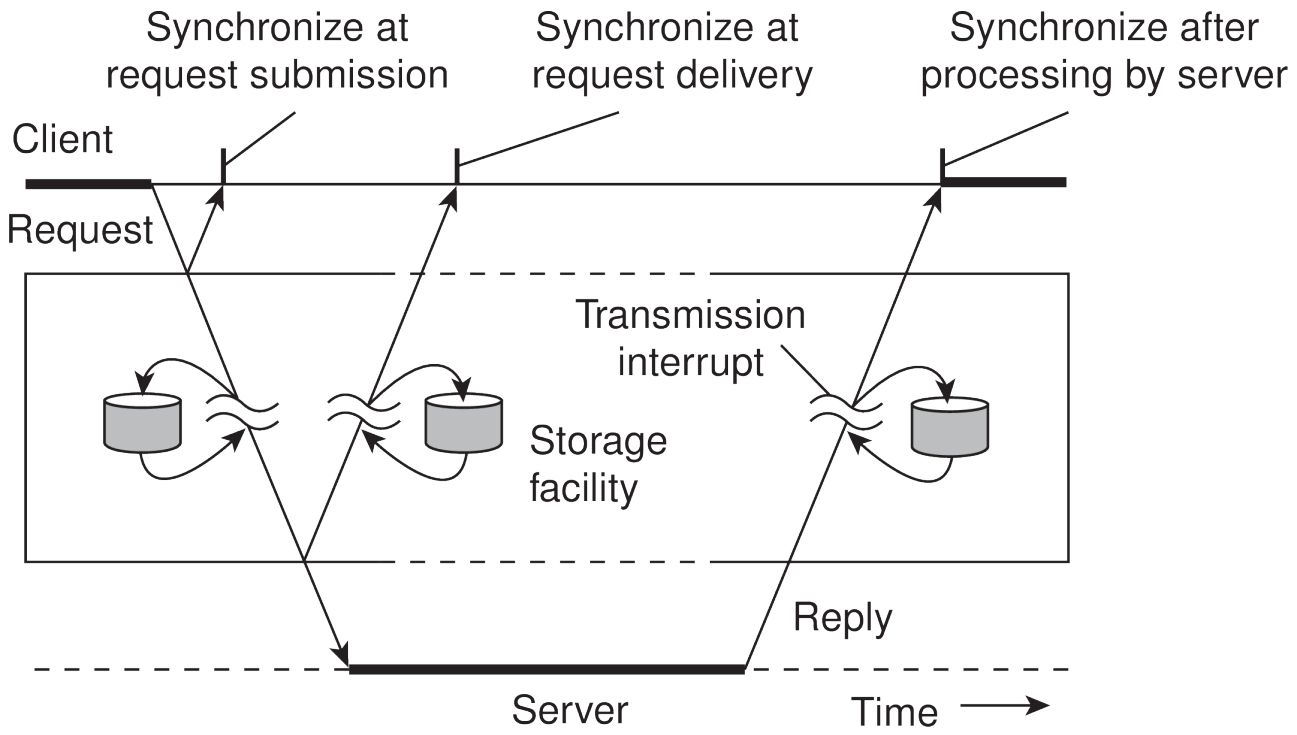
Types of communication (Transient versus persistent)



Maarten van Steen and Andrew S. Tanenbaum, *Distributed Systems*, Fourth edition, Version 4.02, February 2024.

AASTU

Types of communication (Places for synchronization)



Maarten van Steen and Andrew S. Tanenbaum, *Distributed Systems*, Fourth edition, Version 4.02, February 2024.

Client/Server Model

Key Idea

- Based on **transient synchronous communication**.
- Both client and server must be active at the same time.
- Client sends request $\rightarrow$ blocks until reply is received.

Example

- ⋆ **Web Browsing:** Browser (client) sends HTTP request to server, waits for the page before continuing.

AASTU

Drawbacks of Synchronous Client/Server

Limitations

- Client is blocked and cannot do other work while waiting.
- Failures must be handled immediately (timeouts, retries).
- Not suitable for some services (e.g., email, news delivery).

Practical Example

- ⋆ **Online Payment:** If the payment server is slow or crashes, the client app "freezes" until it times out.

AASTU

Remote Procedure Call (RPC): Observations & Conclusion

Observations

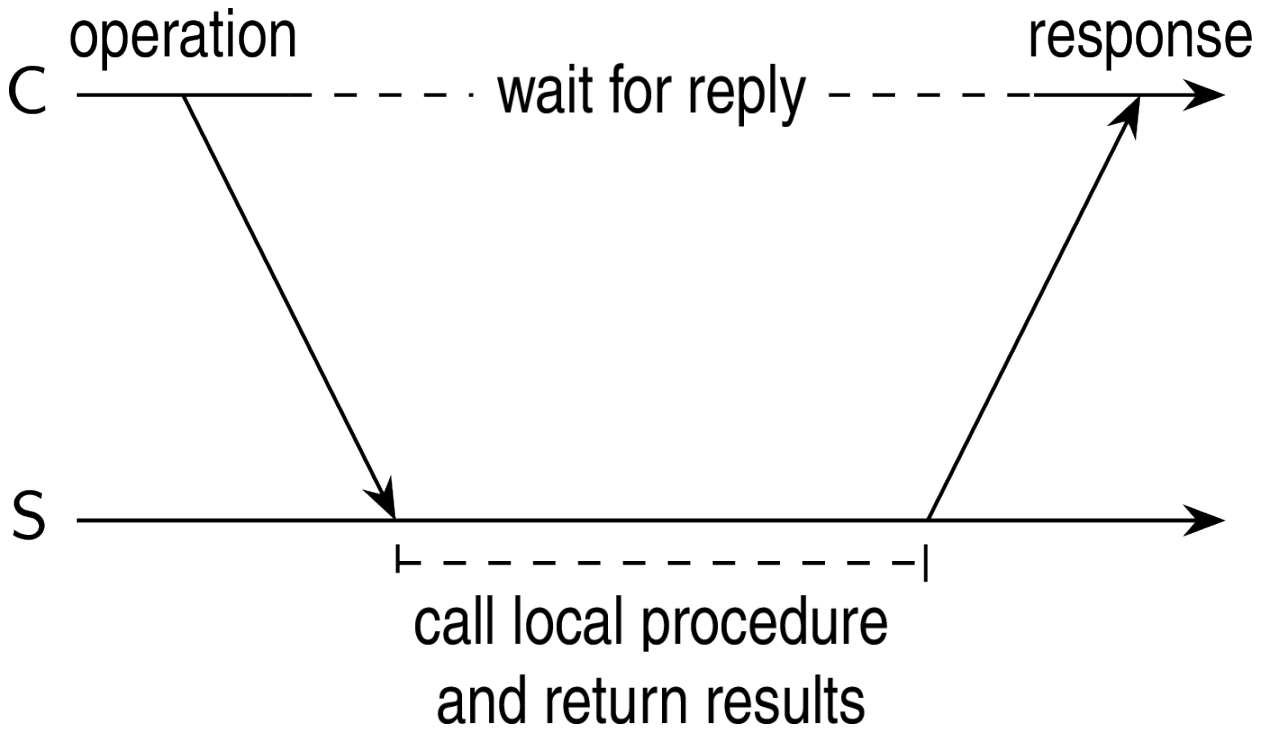
- Developers already use a simple **procedure-call model**.
- Procedures act as **black boxes**, isolated from each other.

Conclusion

- Communication between caller and callee can be **hidden** using procedure calls.
- ⋆ **Example:** A banking app calls `checkBalance()` — it may execute locally or on a remote server, but looks the same to the developer.

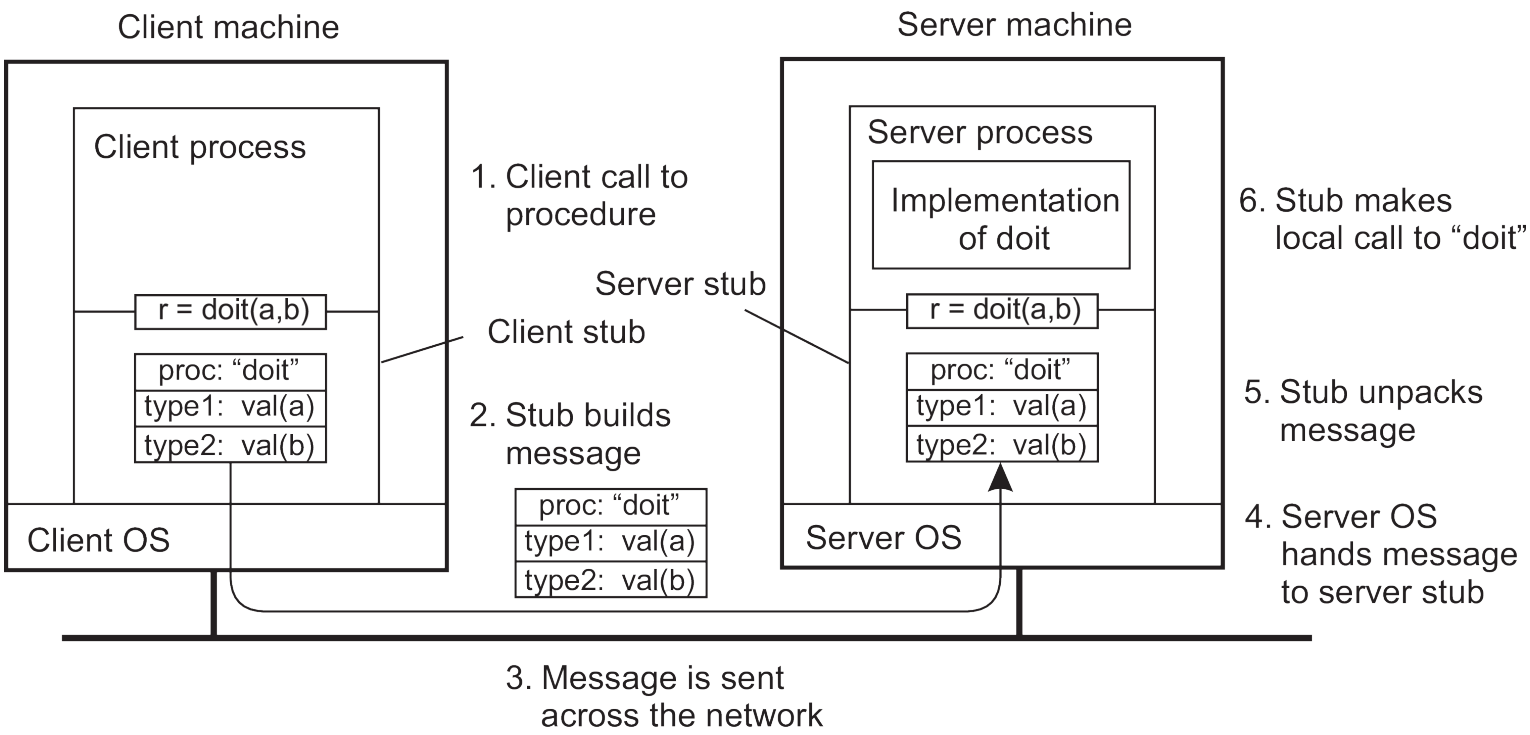
AASTU

Basic RPC operation (Observations)



Maarten van Steen and Andrew S. Tanenbaum, *Distributed Systems*, Fourth edition, Version 4.02, February 2024.

Basic RPC operation



Maarten van Steen and Andrew S. Tanenbaum, *Distributed Systems*, Fourth edition, Version 4.02, February 2024.

AASTU

RPC Operation Steps: Client to Server

Steps 1–5

- Step 1: Client procedure calls the client stub.
- Step 2: Stub builds the message and calls the local OS.
- Step 3: Local OS sends the message over the network.
- Step 4: Remote OS receives the message and passes it to the server stub.
- Step 5: Server stub unpacks parameters and calls the server procedure.

AASTU

RPC Operation Steps: Server to Client

Steps 6–10

- Step 6: Server executes procedure and sends result to stub.
- Step 7: Stub packages response and calls OS.
- Step 8: Server OS transmits message to client OS.
- Step 9: Client OS delivers message to client stub.
- Step 10: Client stub unpacks result and returns it to client.

AASTU

RPC: Parameter Passing

Key Points

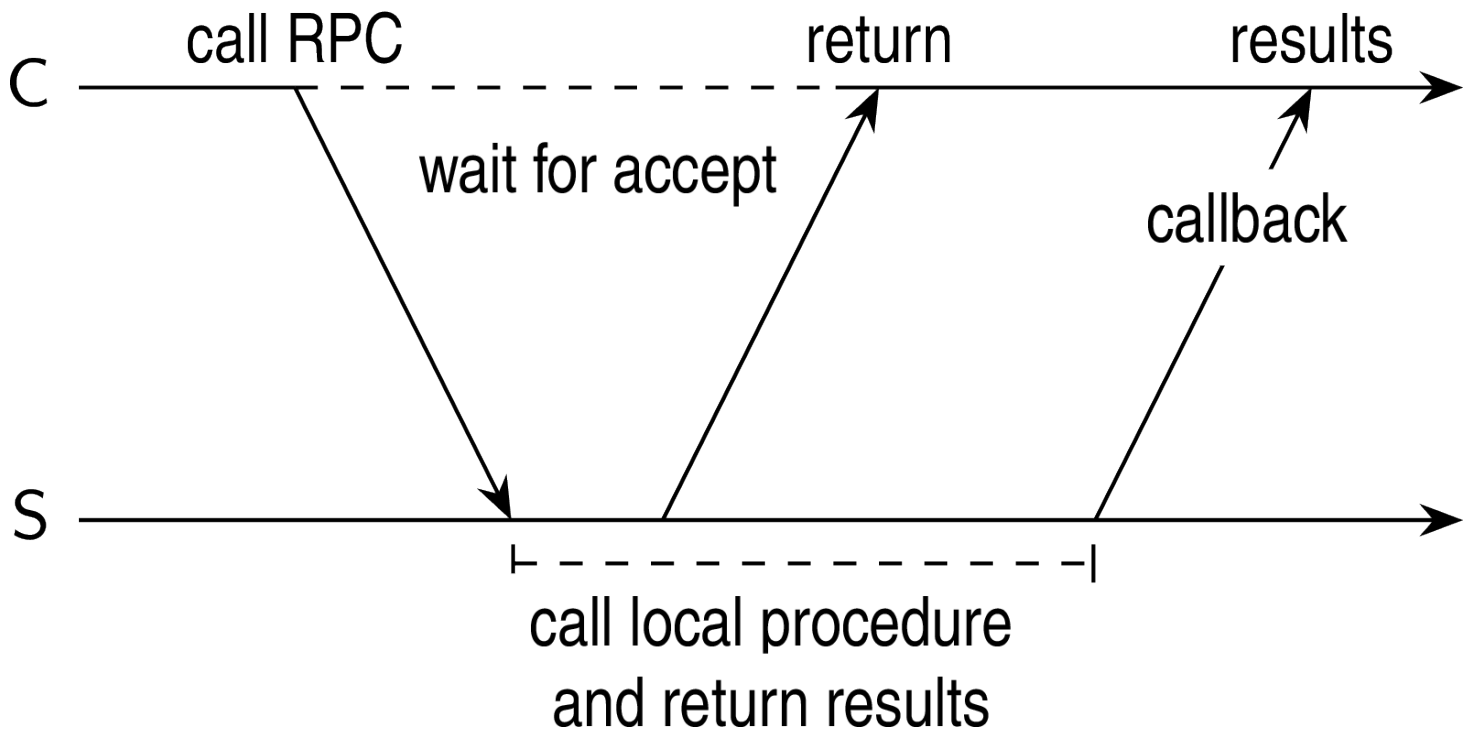
- Client and server may have different data formats.
- Parameters are converted into bytes (marshaling).
- Both must agree on encoding for all data types.

Example

- Sending a float between little-endian and big-endian machines requires consistent encoding.

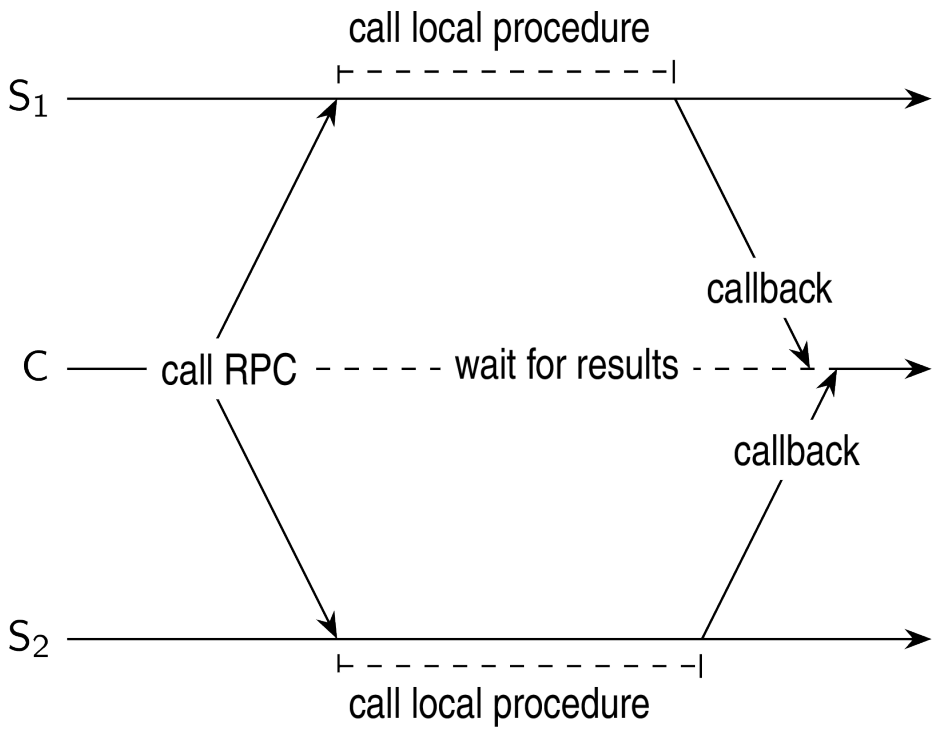
AASTU

Asynchronous RPCs



Maarten van Steen and Andrew S. Tanenbaum, *Distributed Systems*, Fourth edition, Version 4.02, February 2024.

Sending out multiple RPCs



Maarten van Steen and Andrew S. Tanenbaum, *Distributed Systems*, Fourth edition, Version 4.02, February 2024.

Transient messaging: (sockets, Berkeley socket interface)

Common Socket Operations

Operation	**Description**
socket	Create a new communication endpoint.
bind	Attach a local address to a socket.
listen	Tell OS the max number of pending connections.
accept	Block caller until a connection request arrives.
connect	Actively attempt to establish a connection.
send	Send data over the connection.
receive	Receive data over the connection.
close	Release the connection.

AASTU

Sockets: Server

```python
import socket
server_socket = socket.socket(socket.AF_INET,
                              socket.SOCK_STREAM)
server_socket.bind(('localhost', 12345))
server_socket.listen(1)
print("Server listening on port 12345...")
conn, addr = server_socket.accept()
print(f"Connected by {addr}")
data = conn.recv(1024)
print("Received from client:", data.decode())
conn.sendall(b"Hello from server")
conn.close()
print("Connection closed")
```

AASTU

Sockets: Client

```python
import socket
client_socket = socket.socket(socket.AF_INET,
                              socket.SOCK_STREAM)
client_socket.connect(('localhost', 12345))
print("Connected to server...")
client_socket.sendall(b"Hello from client")
data = client_socket.recv(1024)
print("Received from server:", data.decode())
client_socket.close()
print("Connection closed")
```

AASTU

Making Sockets Easier to Work With

Key Points

- Sockets are low-level and prone to programming errors.
- Common usage patterns exist (e.g., client-server).
- ZeroMQ provides a higher-level abstraction for messaging.

Example

- ZeroMQ pairs a sending socket at process P with a receiving socket at process Q.

AASTU

Making Sockets Easier to Work With

Communication Patterns

- Request-Reply: Client sends request, server replies.
- Publish-Subscribe: Publisher sends updates to subscribers asynchronously.
- Pipeline: Tasks distributed among workers in a stream.

Example

- Publish-Subscribe: Server pushes updates to all subscribers.

AASTU

Multicast Communication

Key Points

- Multicast: sending data to multiple receivers simultaneously.
- Historically handled at network/transport layer; setup often complex.
- Peer-to-peer and overlay networks simplify application-level multicasting.
- Gossip-based dissemination: simple, but less efficient alternative.

AASTU

Application-Level Tree Multicasting

Key Points

- Nodes form an overlay network to forward messages.
- Tree vs Mesh:
  - Tree: unique path between nodes.
  - Mesh: multiple paths, more robust to failures.
- Forwarders propagate messages; children receive them.
- Performance metrics: link stress, stretch, tree cost.

AASTU

Overlay Performance: Tree Efficiency (Part 1)

Key Points

- Building a multicast tree is simple; building an efficient one is harder.
- Node selection based only on logical routing may traverse some physical links multiple times.

Example

- Multicast from node A to B $\rightarrow$ E $\rightarrow$ D $\rightarrow$ C crosses physical links: A $\rightarrow$ Ra $\rightarrow$ Rb $\rightarrow$ B $\rightarrow$ Rb $\rightarrow$ Ra $\rightarrow$ Re $\rightarrow$ E $\rightarrow$ Re $\rightarrow$ Rc $\rightarrow$ Rd $\rightarrow$ D $\rightarrow$ Rd $\rightarrow$ Rc $\rightarrow$ C
- Some links are traversed twice, increasing latency and network load.

AASTU

Overlay Performance: Tree Efficiency (Part 2)

Observation

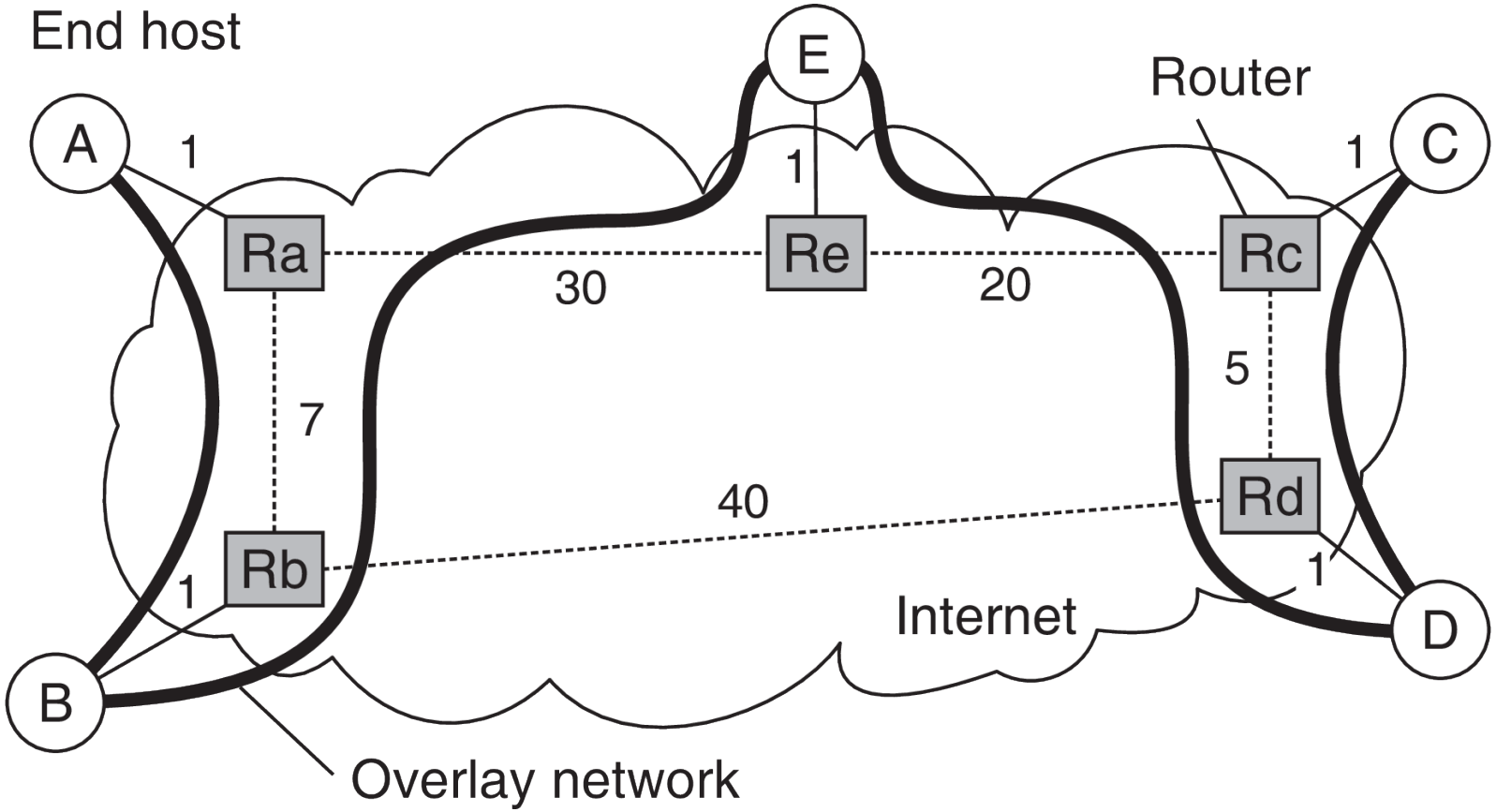
- Redesigning overlay links can improve efficiency.
- Example: Instead of $\langle B, E \rangle$ and $\langle D, E \rangle$, use $\langle A, E \rangle$ and $\langle C, E \rangle$ to reduce double traversal.

Practical Impact

- Reduces total network traversal cost.
- Lowers message delivery delay and avoids congestion on heavily used links, Improves performance metrics for overlay multicast.

AASTU

ALM: Some costs



Maarten van Steen and
Andrew S. Tanenbaum, *Distributed Systems*,
Fourth edition, Version 4.02, February 2024.

Flooding: Key Points

Key Points

- Node P sends message `m` to all neighbors.
- Each neighbor forwards `m` to its neighbors, except the sender, only if it has not seen `m` before.Ensures message reaches all nodes in the network.

Note

Flooding is simple but may generate redundant messages and high network traffic.

AASTU

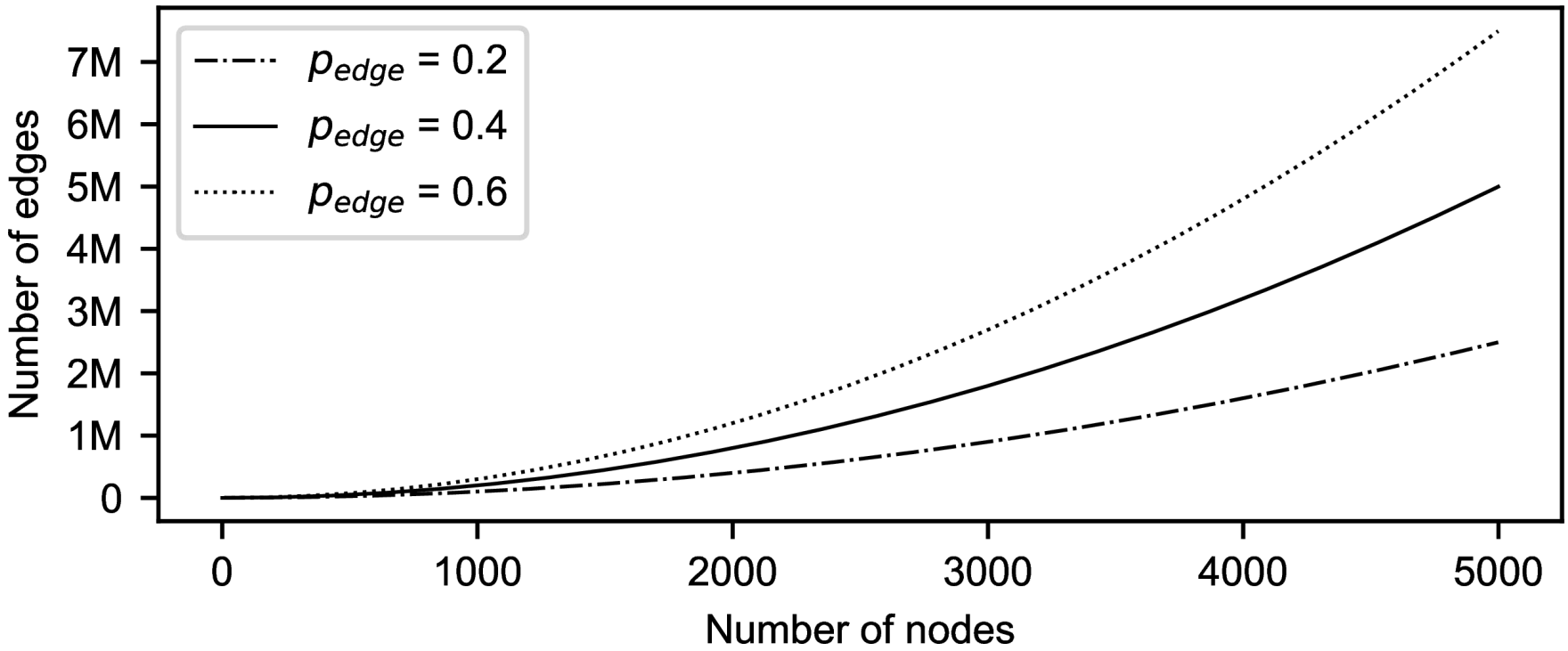
Flooding: Example

Propagation Example

- Node A sends update `m` to B and C.
- B forwards `m` to its neighbors (except A), e.g., D and E.
- C forwards `m` to its neighbors (except A), e.g., F and G.
- Each node only forwards unseen messages, preventing infinite loops.

AASTU

Flooding



Maarten van Steen and
Andrew S. Tanenbaum, *Distributed Systems*,
Fourth edition, Version 4.02, February 2024.

AASTU

Questions?

Thank you for your attention!

AASTU

References

Suggested References

1. M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., Version 4.02, February 2024.
2. G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*, 5th ed., Addison-Wesley, 2012.
3. N. Lynch, *Distributed Algorithms*, 2nd ed., Morgan Kaufmann, 1996.
4. R. E. Bryant and D. O'Hallaron, *Computer Systems: A Programmer's Perspective*, 4th ed., Pearson, 2024.

AASTU