



# DISTRIBUTED COMPUTING

Weekly Summary

3



School of Computing

# PERFORMANCE CONSIDERATIONS IN DISTRIBUTED SYSTEMS

## LATENCY

The delay between sending and receiving data, which can significantly affect performance. High latency can occur due to network congestion or geographical distance.

## BANDWIDTH

Refers to the maximum rate of data transfer. A higher bandwidth is essential for high-demand applications like video streaming or large file transfers.

## THROUGHPUT

The actual rate at which data is transmitted across the system, influenced by network conditions and protocol overhead.

## RELIABILITY AND FAULT TOLERANCE

Ensuring system reliability by implementing redundancy, error handling, and reliable messaging protocols to manage potential failures.

## STREAM-ORIENTED COMMUNICATION

Stream-oriented communication allows continuous data flow between processes in real-time, which is crucial for time-sensitive applications such as video conferencing and live financial transactions.

**Low overhead:** Stream-oriented communication typically involves less protocol overhead compared to message-based communication, allowing more efficient real-time data transmission.

**Bidirectional communication:** Some stream-oriented communication models, like video calls, allow two-way communication where both parties can send and receive data simultaneously.

**Error handling:** Built-in error recovery mechanisms ensure data integrity during transmission, even in cases of packet loss or transmission errors.

# TYPES OF STREAM-ORIENTED COMMUNICATION

## TCP STREAMS

Offers reliable, ordered, and error-checked communication, widely used for applications that require consistent data delivery.

## WEBSOCKETS

Full-duplex communication channels ideal for real-time applications like online gaming and messaging.

## UDP STREAMS

Used for applications like VoIP and online gaming where speed is more critical than reliability.

## INTER-PROCESS COMMUNICATION (IPC) IN DISTRIBUTED SYSTEMS

### MESSAGE PASSING

A communication model where processes send explicit messages to each other. Suitable for independent processes in different machines or environments.

### REMOTE PROCEDURE CALL (RPC)

RPC allows a program to execute a procedure on a remote system, making remote calls appear local.

### REMOTE METHOD INVOCATION (RMI)

Java-specific mechanism allowing methods on remote objects to be called seamlessly, especially within Java ecosystems.

## KEY DIFFERENCES BETWEEN RPC AND RMI

1

2

### RPC

Suitable for language-neutral, system-independent communication. It does not support object serialization but is efficient in networked environments.

### RMI

Java-specific and supports the passing of objects between systems. It has more overhead due to object serialization but is well-integrated within Java-based applications.

## USE CASES FOR RPC

**Banking Systems:** For fast, language-neutral transfers between branches using synchronous calls.

**Distributed File Systems:** For accessing files over a network where efficient, lightweight communication is required.

## USE CASES FOR RMI

**Online Shopping Systems:** RMI suits Java ecosystems for managing complex, object-rich communication like shopping carts and inventory systems.

**Chat Applications:** Suited for exchanging message objects between users in real-time.

## PERFORMANCE CONSIDERATIONS IN RPC AND RMI

### RPC

More efficient with lower overhead, especially suitable for real-time systems requiring rapid communication.

### RMI

Higher CPU usage due to object serialization and more suitable for applications where object passing is essential.

## FAULT TOLERANCE IN IPC

Techniques such as retries, timeouts, and logging are used to ensure reliability and recovery from failures.

## SECURITY IN IPC



### AUTHENTICATION

Verifies the identity of clients and servers.



### ENCRYPTION

Protects the confidentiality of the messages being exchanged.



### AUTHORISATION

Controls access permissions for processes or data.

## TOOLS FOR RPC

gRPC: A high-performance framework for RPC.

ONC RPC: Traditional UNIX-based RPC tool.

Apache Thrift: A framework for cross-language RPC.

## TOOLS FOR RMI

Java RMI Registry: Locates and binds remote objects.

rmic: Tool to generate stubs and skeletons for RMI.

Policy Files: Control access permissions for RMI services.

## REFLECTION QUESTIONS

- What are the primary differences between RPC and RMI in terms of use cases and efficiency?
- How do stream-oriented communication models differ from message-based models in terms of performance and real-time data transmission?
- When would you choose RPC over RMI, and why?
- What role does latency play in choosing between TCP, WebSockets, and UDP for stream-oriented communication?
- How can you address performance issues related to bandwidth and latency in stream-oriented communication for applications like live video streaming?

## CALL TO ACTION

Review and compare RPC and RMI mechanisms to understand which one best suits your distributed system's needs.

Reflect on real-world applications such as online banking and shopping to understand when to implement RPC or RMI, based on system requirements and scalability.

# QUESTIONS TO PONDER ON

- How do you think object serialization in RMI impacts system performance compared to the function call-based RPC?
- What steps can be taken to optimize the performance of RPC in high-latency environments?
- How does the continuous flow of data in stream-oriented communication impact network design and optimization?
- What challenges might arise when integrating RPC into systems that require cross-platform communication?
- How can security mechanisms like encryption and authentication be integrated into RPC and RMI to ensure safe data exchange in distributed systems?

## SKILLS AND COMPETENCIES ACQUIRED AFTER THIS LESSON

-  SYSTEM PERFORMANCE EVALUATION
-  FAULT TOLERANCE DESIGN
-  NETWORK LATENCY MANAGEMENT
-  REAL-TIME COMMUNICATION HANDLING
-  INTER-PROCESS COMMUNICATION DESIGN
-  RPC AND RMI IMPLEMENTATION
-  DATA SERIALIZATION AND OBJECT PASSING
-  SECURITY AND AUTHENTICATION IN DISTRIBUTED SYSTEMS