CMPSCI 677 Distributed and Operating Systems **Spring 2025**

Lecture 3: February 10

Lecturer: Prashant Shenoy
Scribe: Angela Zhu (2025), Sai Satvik Vuppala (2024), Shreya Dubey (2023), Rajul Jain (2022)

3.1 Communication in Distributed Systems

3.1.1 Communication between processes

Suppose two processes running on same machine want to communicate with each other. There are 2 ways to achieve this.

Unstructured Communication:

- Processes use a shared memory/buffer or a shared data structure for communication.
- This type of communication works better if the processes reside on the same system.
- This type of communication is called "unstructured" because the buffer is just a piece of memory and has no structure associated with it. In other words, we can put any data that we want in the buffer and the processes have to interpret that data and do something with it.

Structured Communication:

- Processes send explicit messages to communicate, i.e., using Inter-Process Communication (IPC).
- This type of communication works regardless of the processes being on the same machine.
- This can be achieved by either low-level socket-based message passing or higher-level remote procedure calls.

Now suppose the two processes are running on different machines in a distributed system. If the processes communicate via unstructured communication, then we need to figure out a way to share buffers across machines. Clearly, making memory accessible across machines connected over a network is more difficult than using shared memory in the same machine. Hence, we prefer structured communication like sockets and RPCs.

Question (Student): Why do distributed systems need low-level communication support for both structured and unstructured communication?

Answer (Instructor): Because distributed systems are spread over a network.

3.1.2 Communication (Network) Protocols

These serve as the established guidelines for communication between processes, facilitating a shared language for inter-process communication. Suppose two processes (App 1 and App 2), each using the OSI or TCP/IP model, communicate over a network. App 1 creates a message at the application layer which travels down the network stack layers (where each layer adds their headers to the original message) all the way to the physical layer. The message is then passed over to App 2's physical layer where it travels all the way up the network stack again to complete the communication.

Question (Student): Can the ordering of the layers change?

Answer (Instructor): The ordering will not change.

Question: (Student): Can the encryption happen at a higher level?

Answer: (Instructor): Only the message can be encrypted, not the other aspects. If enrcypting after Data link layer, nobody can be see what is there in it. However, hardware can be used to encrypt the message itself.

3.1.3 Middleware Protocols

In a distributed system, *middleware* is the layer residing between the OS and an application. The middleware layer sits between the application and transport layer in the network protocol stack.

3.1.4 TCP-based Socket Communication

TCP-based **socket communication** is a structured communication method which uses TCP/IP protocols to send messages between processes on different machines. The socket creates network address (IP address and port number) that becomes the channel for communication. The socket interface creates network end-points. For example, in a client-server distributed system, both the client and the server need to create sockets which are bound to port number where they can listen to and send messages.

Question (Student) : What is the difference between a socket and a port? And what is a socket?

Answer: (Instructor): A port number is the identifier of a socket and a socket is an abstraction for network communication. For example, a file is an abstraction for stored data that is read and written to a disk. A socket is an abstraction that allows you to communicate with other processes by sending and receiving messages.

Question (Student) : How many users can connect to a port?

Answer: (Instructor): Any number of users can connect to a port.

Question (Student) : Every time a client connects, is there a socket connection?

Answer: (Instructor): Yes, every client has its own TCP/IP connection to the server.

Question (Student) : What happens if the same client has multiple requests?

Answer: (Instructor): Depends on how the client is implemented. It can be a sequential client if it's in a while loop sending one request at a time or if the client has multiple threads, they can open multiple TCP connections to the server and send requests in parallel on each of these connections.

Question (Student) : How many processes can listen on the same port number?

Answer: (Instructor): There can be only on socket that is associated with a socket of a certain port number. For example, you can have multiple web servers but these should listen on different port numbers.

Question (Student) : How does this work for P2P (Peer-to-Peer) communication happen on the socket level?

Answer: (Instructor): Essentially, you should think of it as processes A and B that want to communicate. In the Peer-to-Peer case, one peer runs the client code and the other will be waiting for other peers to send it messages, so for the purposes of communication it will be running the server side code.

Question (Student) : What does it mean to listen? And what does it do?

Answer: (Instructor): In this case, Listen tells the operating system (OS) that the socket is ready to receive messages. The socket and bind calls just prepare the socket. Once you are ready to start communication, you will make the listen call which means the server is ready to wait for messages. Once a message is received, the server will accept the message. The server will sit in an infinite loop of listening.

3.1.4.1 Understanding TCP Network Overheads

Normal Operation of TCP: Consider a client-server model using TCP method for communication. As TCP method follows a 3-way handshake protocol for communication, this leads to a transfer of 9 messages for sending a single request and subsequently receiving a single response. This is a huge overhead. For a single request and single response this is a high overhead but if there is lots of communication between the client and the server, these extra 9 messages wouldn't be a huge deal.

Transactional TCP: There were efforts made to reduce the overhead. Transactional TCP is a protocol that look like TCP. Instead of sending 9 messages as in the above case, here we batch up the messages to reduce the total number of messages going to and fro. This is not used in practice, generally because both endpoints have to follow the same rules and this ends up not being used as often.

Question (Student) : What happens when there are multiple client requests?

Answer: (Instructor): Typically, it depends on the protocol. In sequential processing, the client sends a request and then waits for an answer. There can be multiple requests with the same connection. You can send a request before the response to the first request is received only in some protocols. This requires tracking which question this response corresponds to.

Question (Student) : What happens if the client doesn't receive the response?

Answer: (Instructor): All the messages have a timeout. If the acknowledgment is not received by a certain time, the request will timeout. It is assumed that the message after the previous message was lost so the client sends it again.

Question (Student) : Is a separate acknowledgement required for each message?

Answer: (Instructor): Packets are sent in sequence, so if acknowledgement for a message is sent, then it means all the messages before it were received, so you can send just one acknowledgement for multiple messages.

Question (Student) : Why does the server send the SYN message?

Answer: (Instructor): SYN message is used to establish one-way connection. Through the first SYN, the client establishes a one-way connection to the server. Only client can send messages to the server. In order to send messages to the client, the server needs to send a SYN message to establish one-way connection to the client. This will then form a duplex connection.

Question (Student) : When will the server refuse to connect with a client?

Answer: (Instructor): As long as the socket on which it is listening is open, it can continue to receive new connections. The only way to stop is to close the conection.

Question (Student) : What does teardown the connection mean?

Answer: (Instructor): Similar to closing the file, so that we cannot read or write to it.

Question (Student) : Reason why transactional TCP didn't get implemented?

Answer: (Instructor): To implement any change to any network protocol every computer in the world needs to be changed. Normal TCP is widely used, there is no drawback to transactional TCP, it is just hard to deploy in the real world.

3.1.5 Group Communication

In a distributed system, when one machine needs to communicate with many machines, group communication protocols are used. This is analogous to sending an email to multiple recipients. For group communication, all the recipients subscribe to a *group address.* The network then takes care of delivering the messages to all the machines in the group address. This is called *multicasting.* If the messages are sent to all the machines in the network, then the process is called *broadcasting.* Not all machines have multicast capabilities. If we want to do multicast regardless of the underlying hardware we have to do some processing at the application layer. This is basically a library at the application layer that implements multicast as multiple unicast (one to one) messages.

Question (Student) : Who resolves the group address?

Answer: (Instructor): To implement group communication, you need additional network support. Ensuring the messages are delivered is typically done in the routing layer. The routers need to have support turned on for group communication, if it is not turned on it will just be ignored.

3.2 Remote Procedure Calls (RPC)

RPCs provide higher level abstractions by making distributed computing look like centralized computing. They automatically determine how to pass messages at the socket level without requiring the programmer to directly implement the socket code. In other words, *instead of sending a message to the server to invoke a method X, the programmer can call the method X directly from the client machine.* RPCs are built using sockets underneath. The programmers do not write socket code, instead, it is auto-generated by the RPC compiler. Stubs are another piece of RPC compiler auto-generated code which convert parameters passed between client and server during RPC calls.

There are a few semantics to keep in mind:

- Calling a method using RPC is same as invoking a local procedure call.
- One difference is that in an RPC, the process has to wait for the network communication to return before it can continue its execution, i.e., RPCs are *synchronous.*
- *You cannot pass pointers or references.* Pointers and references point to a memory location in the respective machine. If these memory locations are passed on a different machine, they will point to something completely different on that machine.
- *You cannot pass global variables.* As global variables lying on one machine cannot be accessed by another machine simply over a network. Hence global variables are not allowed in an RPC.
- *Pass arguments by value.*

- *Differences between client/server.* The client and server may not be homogeneous. You need to be careful about the platform each machine is running on. The machines might be running different OS (Windows, Mac). You cannot make assumptions of what is running on the other end.

Question: (Student) : If we absolutely need to pass pointers in an RPC, how do we achieve that?

Answer: (Instructor): Take the entire object (say, from the client machine) and pass it on to the server. Create a copy of this object on server and then create a local pointer to the object.

Question (Student) : Does the connection stay open from the client side?

Answer: (Instructor): Depends on the stub code. It can decide to setup a new TCP/IP connection for every RPC or send multiple RPCs on the same connection.

Question (Student) : When we write a message from the client, do we need to know the format of the message?

Answer: (Instructor): You don't write a message but invoke a function. Thus, only the function call needs to be constructed with the correct parameters which is then sent as a function call. The rest is taken care of by the stub.

Question (Student) : Any real world application of RPCs?

Answer: (Instructor): They are widely used for all kinds of applications. Many servers use RPCs, for example file servers. There are many RPC implementations as well, in the next lecture we will talk about Google's implementation of an RPC called gRPC.

Question (Student) : When is RPC preferred over TCP?

Answer: (Instructor): All communication is done through TCP / IP. With RPC the programmer did not have to write TCP/IP code for this communication, but it is built on top of sockets and uses network communication.

Question (Student) : What is the difference between RPC and REST APIs?

Answer: (Instructor): RPCs are one layer below things like REST. REST is using HTTP for communication, RPC does not necessarily need to use HTTP there are many communication methods it can use. RPCs are using procedures as an abstraction for inter-process communication. You can think of it as REST APIs are a form remote procedure call but you are using HTTP as your mode of transport, ie. HTTP based RPC.

3.2.1 Parameter Passing

There are two ways to pass arguments there is pass by value and pass by reference. By value you will be making a copy of that object and send the copy. Whereas when you pass by reference you will be passing the address to the object. And therefore, you can actually modify the original object.

3.2.2 Marshalling and Unmarshalling

Different machines use different representation of data formats.This creates discrepancy in understanding messages and data between different machines. Hence before sending messages to the other machine, the messages are converted into a standard representation such as eXternal Data Representation (XDR). This is how complex data structures are sent and restructured in different machines. Marshalling is the process of converting data on one machine into a standard representation suitable for storage/transmission. Unmarshalling is the process of converting from a standard representation to an internal data structure understandable by the respective machine.

Question (Student) : What is the advantage of using RPC over HTTP?

Answer: (Instructor): HTTP is an application-level protocol used for sending and getting replies. But RPCs allow any two applications to communicate with each other.

3.2.3 Binding

The client locates the server using bindings. The server that provides the RPC service registers with a naming/directory service and provides all of the details such as method names, version number, unique identifier, etc. When client needs to access a specific method/functionality, it will search in the naming/directory service if there's a service in the network which hosts this method or not and then accesses the server using the IP and port number listed in the directory.

Question (Student) : What if there are two servers/functions with the same name?

Answer: (Instructor): To avoid collisions we need to make sure that the name is unique, we can just add a version number to the name to make it unique.

3.3 RPC Implementation

3.3.1 Failure semantics

- Lost request/ response messages: The network protocols handle these errors by timeout mechanisms.
- Server and client failures: Need to be handled while creating distributed systems.

If client crashes after sending a request to the server, then the server response is referred to as **orphan**. To deal with the orphans, methods such as Extermination, Reincarnation, Gentle reincarnation, and expiration can be used.

3.3.2 Case study: SUNRPC

SUNRPC was developed for use with NFS. It is one of the widely used RPC systems. Initially, it was built on top of UDP, but later transferred to TCP. It uses SUN's XDR format.

Questions on Failure semantics:

Question (Student) : What if the replies from the server were lost?
Answer: (Instructor): The client did not receive a response here so it will send a request again, this will be a problem when the server is making stateful responses - it will give an error saying the same request is being processed again. Solution is to make all the operations idempotent, irrespective of how many times a request is sent, it will be executed only once and the same response will be sent back every time.

Question (Student) : How does a system become idempotent?
Answer: (Instructor): This is a server side issue, if the server receives same request again it should execute it only once. The client is unaware what is happening at the client side.

Question (Student) : For example: Every time a person makes a payment, it generates a new transaction id, in this case how does the server know if its the same request? (here the server is idempotent)
Answer: (Instructor): The application will re-try with the same rpc request with the same transaction-id.