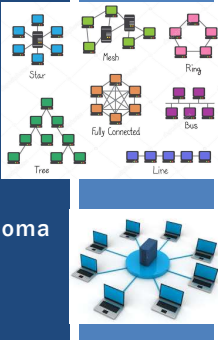


TCCS 558: APPLIED DISTRIBUTED COMPUTING

Communication

Wes J. Lloyd
 Institute of Technology
 University of Washington - Tacoma



OBJECTIVES

- Assignment 1 questions
- The role for UDP
- Ch. 4 – Communications
 - Protocols
 - Remote procedure calls / RMI
 - Message-oriented communication:
 - sockets, zeromq, MPI

November 9, 2017
TCCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma
L12.2

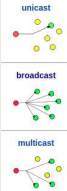
CONNECTIONLESS / CONNECTION-ORIENTED

- **Connection-oriented communication (TCP)**
 - Two parties connect, exchange messages, and the disconnect
 - Typically this is a synchronous process, but it can be asynchronous
- **Connectionless communication (UDP)**
 - Calling program does not enter into a connection with the target process
 - Receiving application simply acts on the request
 - This may, or may not, involve sending a response

November 9, 2017
TCCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma
L12.3

WHAT ARE USE CASES FOR UDP?

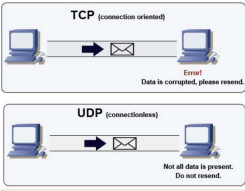
- Processes/applications that already provide:
 - Internal flow control (packet ordering)
 - Error control (management of retransmission requests)
- Broadcasting (sending to subnet)
- Multicasting (addressing to multiple clients)
 - Typically in a LAN
- Simple request-response communication
 - UDP makes sense for really small transactions because there is no TCP establishment/tear-down overhead
 - Latency is reduced: one-way trip, or out-and-back, but no negotiation
 - Bandwidth user: When total communication is less than MTU
 - **Maximum Transmission Unit**: < largest packet size (~1500 avg)



November 9, 2017
TCCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma
L12.4

UDP USE CASES

- When overhead for creating a TCP connection far outweighs data payload
 - DNS servers (quick negotiation of names)
 - Network Time servers
 - Service discovery (via LAN broadcast): finding a printer
 - When delivering data that CAN be lost without consequence because newer data is always flowing in to replace previous state
 - Weather data, video/audio (VoIP) streaming, video gaming data



November 9, 2017
TCCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma
L12.5

UDP USE CASES - 2

- UDP can be used for every type of application TCP can
- Requires implementation of proper retransmission mechanism.
- UDP can be very fast, with low delay, not affected by congestion on a connection basis, transmits fixed sized datagrams and can be used for multicasting.
- *If implementing an application level protocol . . .*
- **What would the advantages be for using UDP ?**
- **What would the advantages be for using TCP ?**

November 9, 2017
TCCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma
L12.6

CONNECTIONLESS / CONNECTION-ORIENTED - 2

- A component interacts requests to establish a subscription to receive notifications regarding particular data from a *shared "tuple" data space*
 - **IS THIS: Connection-less or connection oriented?**
- Components publish data to a *shared "tuple" data space*
 - **IS THIS: Connection-less or connection oriented?**

Shared (persistent) data space

November 9, 2017 TCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma L12.7

CHAPTER 4

- 4.1 Foundations
 - Protocols
 - Types of communication
- 4.2 Remote procedure call
- 4.3 Message-oriented communication
 - Socket communication
 - Messaging libraries
 - Message-Passing Interface (MPI)
 - Message-queueing systems
 - Examples
- 4.4 Multicast communication
 - Flooding-based multicasting
 - Gossip-based data dissemination

November 9, 2017 TCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma L12.8

CH. 4.1: FOUNDATIONS

L12.9

MIDDLEWARE PROTOCOLS

- Communication frameworks/libraries
- Reused by multiple applications
- Provided needed functions apps build and depend on
- Example:
 - **Authentication protocols:** supports granting users and processes access to authorized resources
 - General, application-independent in nature
 - Doesn't fit as an "application specific" protocol
 - Considered as a "Middleware protocol"

November 9, 2017 TCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma L12.10

MIDDLEWARE PROTOCOLS - 2

- **Distributed commit protocols**
 - Coordinate a group of processes (nodes)
 - Facilitate all nodes carrying out a particular operation
 - Or abort transaction
 - Provides distributed atomicity (all-or-nothing) operations
- **Distributed locking protocols**
 - Protect a resource from simultaneous access from multiple nodes
- **Remote procedure call**
 - One of the oldest middleware protocols
 - Distributed objects

November 9, 2017 TCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma L12.11

MIDDLEWARE PROTOCOLS - 3

- **Message queueing services**
 - Support synchronization of data streams
 - Transfer real-time data
 - Distributed and scalable implementation
- **Multicast services**
 - Scale communication to thousands of receivers spread across the Internet

November 9, 2017 TCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma L12.12

ADAPTED REFERENCE MODEL

- Shows layers actually used

Network

Combines network and transport
Physical and Data link

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma L12.13

TYPES OF COMMUNICATION

- Persistent communication**
 - Message submitted for transmission is stored by communication middleware as long as it takes to deliver it
 - Example: email system (SMTP)
 - Receiver can be offline when message sent
 - Temporal decoupling (delayed message delivery)
- Transient communication**
 - Message stored by middleware only as long as sender/receiver applications are running
 - If recipient is not active, message is dropped
 - Transport level protocols typically are transient (no msg storage)

At what reference model layer is the SMTP Protocol?
From an implementation point-of-view what major component is required to implement persistent communication?

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma L12.14

TYPES OF COMMUNICATION - 2

- Asynchronous communication**
 - Client does not block, continues doing other work
- Synchronous communication**
 - Client blocks and waits
- Three types of **blocking**
 - Until middleware notifies it will take over delivering request
 - Sender may synchronize until request has been delivered
 - Sender waits until request is processed and result is returned
- Persistence + synchronization**
 - Common scheme for message-queueing systems

Consider each type of blocking (1, 2, 3). Are these modes connectionless (UDP)? connection-oriented (TCP)?

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma L12.15

CH. 4.2: RPC

L12.16

RPC - REMOTE PROCEDURE CALL

- In a nutshell,
- Allow programs to call procedures on other machines
- Process on **machine A** calls procedure on **machine B**
- Calling process on **machine A** is suspended
- Execution of the called procedure takes place on **machine B**
- Data transported from caller (**A**) to provider (**B**) and back (**A**).
- No message passing is visible to the programmer
- Distribution transparency:** make remote procedure call look like a local one
- `newlist = append(data, dbList)`

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma L12.17

RPC - 2

- Transparency enabled with client and server "stubs"
- Client has "stub" implementation of the server-side function
- Interface exactly same as server side
- But client **DOES NOT HAVE THE IMPLEMENTATION**
- Client stub:** packs parameters into message, sends to server. Calls blocking receive routine and waits for reply
- Server stub:** transforms incoming request into local procedure call
- Server blocks waiting for msg
- Server stub unpacks msg, calls server procedure
- It's as if the routine were called locally**

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma L12.18

RPC - 3

- Server packs procedure results and sends back to client.
- Clients “receive” call unblocks and data is unpacked
- Client can't tell method was called remotely over the network (except when there's HIGH network latency...)
- Call abstraction **allows clients to invoke functions in alternate languages, on different machines**
- Differences are handled by the RPC “framework”

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L12.19

RPC STEPS

1. Client procedure **calls** client stub
2. Client stub **builds** message and calls OS
3. Client's OS **sends** message to remote OS
4. Server OS **gives** message to server stub
5. Server stub **unpacks** parameters, calls server
6. Server **performs** work, **returns** results to server-side stub
7. Server stub **packs** results in messages, **calls** server OS
8. Server OS **sends** message to client's OS
9. Client's OS **delivers** message to client stub
10. Client stub **unpacks** result, returns to client

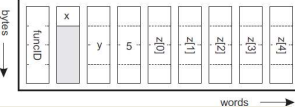
November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L12.20

RPC STEPS

1. Client procedure **calls** client stub
2. Client stub **builds** message and calls OS
- 3.
4. **Consider the overhead of an RPC call vs. an ordinary local procedure call where data elements are pushed/popped, to/from, the call stack**
- 5.
- 6.
- 7.
- 8.
9. Client's OS **delivers** message to client stub
10. Client stub **unpacks** result, returns to client

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L12.21

PARAMETER PASSING



- Stubs: take parameters, pack into message, send across network
- Parameter marshaling:
 - `newlist = append(data, dbList)`
 - Two parameters must be sent over network and correctly interpreted
- Message is transferred as a series of bytes
- Data is serialized into a “stream” of bytes
- Must understand how to unmarshal (deserialize) data
- Processor architecture vary with how bytes are numbered: Intel (right→left), older ARM (left→right)

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L12.22

RPC: BYTE ORDERING

- Big-Endian: write bytes left to right (ARM)
- Little-endian: write bytes right to left (Intel)
- Network: typically transfer data in Big-Endian form
- Solution: transform data to machine/network independent format
- Marshaling/unmarshaling: transform data to neutral format

| BIG-ENDIAN | | | | | | | | Memory | |
|------------|----|-----|-----|-----|-----|-----|-----|--------|-----|
| ... | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | ... |
| | a | a+1 | a+2 | a+3 | a+4 | a+5 | a+6 | a+7 | |

| LITTLE-ENDIAN | | | | | | | | Memory | |
|---------------|----|-----|-----|-----|-----|-----|-----|--------|-----|
| ... | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 | ... |
| | a | a+1 | a+2 | a+3 | a+4 | a+5 | a+6 | a+7 | |

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L12.23

RPC: PASS-BY-REFERENCE

- Passing by value is straightforward
- Passing by reference is **challenging**
- Pointers only make sense on local machine owning the data
- Memory space of client and server are different
- **Solutions to RPC pass-by-reference:**
 1. Forbid pointers altogether
 2. Replace pass-by-reference with pass-by-value
 - Requires transferring entire object/array data over network
 - **Read-only optimization:** don't return data if unchanged on server
 3. Passing global references
 - Example: file handle to file accessible by client and server via shared file system

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L12.24

RPC: DEVELOPMENT SUPPORT

- Let developer specify which routines will be called remotely
 - Automate client/server side stub generation for these routines
- Embed remote procedure calling into the programming language
 - E.g. Java RMI

November 9, 2017
TCS558: Applied Distributed Computing [Fall 2017]
Institute of Technology, University of Washington - Tacoma
L12.25

STUB GENERATION

- `void func(char x; float y; int z[5])`
- Character transmits with 3-padded bytes
- Float as whole word (4-bytes)
 - Array as group of words, proceed by word describing length
 - Client stub must package data in specific format
 - Server stub must receive and unpackage in specific format
- Client and server must agree on representation of simple data structures: int, char, floats w/ little endian
- RPC clients/servers: must agree on protocol
 - TCP? UDP?

November 9, 2017
TCS558: Applied Distributed Computing [Fall 2017]
Institute of Technology, University of Washington - Tacoma
L12.26

STUB GENERATION - 2

- Interfaces often specified using an Interface Definition Language (IDL)
- IDL interface can be used to generate language specific threads
- IDL is compiled into client and server-side stubs
- Much of the plumbing for RPC involves maintaining boilerplate-code

November 9, 2017
TCS558: Applied Distributed Computing [Fall 2017]
Institute of Technology, University of Washington - Tacoma
L12.27

LANGUAGE BASED SUPPORT

- Leads to simpler application development
- Helps with providing access transparency
 - Differences in data representation, and how object is accessed
 - Inter-language parameter passing issues resolved:
→ **Just 1 language**
- Well known example: **Java Remote Method Invocation**
RPC equivalent embedded in Java

November 9, 2017
TCS558: Applied Distributed Computing [Fall 2017]
Institute of Technology, University of Washington - Tacoma
L12.28

RPC VARIATIONS

- RPC: typically client blocks until reply is returned
- Strict blocking **unnecessary** when there is no result
- **Asynchronous RPCs**
 - When no result, server can immediately send reply

Client/server synchronous RPC

Client/server asynchronous RPC

November 9, 2017
TCS558: Applied Distributed Computing [Fall 2017]
Institute of Technology, University of Washington - Tacoma
L12.29

RPC VARIATIONS - 2

- What are tradeoffs for synchronous vs. asynchronous procedure calls?
 - For a local program
 - For a distributed program (system)
- Use cases for asynchronous procedure calls
 - Long running jobs allow client to perform alternate work
 - Client may need to make multiple service calls to multiple server backends at the same time...

November 9, 2017
TCS558: Applied Distributed Computing [Fall 2017]
Institute of Technology, University of Washington - Tacoma
L12.30

TYPES OF ASYNCHRONOUS RPC

- Deferred synchronous RPC**
 - Server performs **CALLBACK** to client
 - Client, upon making call, spawns separate thread which blocks and waits for call

- One-way RPCs**
 - Client **does not wait** for *any* server acknowledgement – it just goes...
- Client polling**
 - Client (*using separate thread*) continually polls server for result

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L12.31

MULTICAST RPC

- Send RPC request *simultaneously* to group of servers
- Hide that multiple servers are involved
- Consideration:**
Does the client need all results or just one?
- Use cases:**
 - Fault tolerance:** wait for just one
 - Replicate execution:** verify results, use first result
 - Divide and conquer:** multiple RPC calls work in parallel on different parts of dataset, client aggregates results

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L12.32

RPC EXAMPLE: DISTRIBUTED COMPUTING ENVIRONMENT (DCE)

- DCE – basis for Microsoft's distributed computing object model (DCOM)
- Used in Samba – share windows filesystem via RPC
- Middleware system:** provides layer of abstraction between OS and distributed applications
- Designed for Unix, ported to all major operating systems
- Install DCE middleware on set of heterogeneous machines – distributed applications can then run and leverage resources
- Uses client/server model
- All communication via RPC
- DCE provides a daemon to track participating machines, ports

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L12.33

DCE – CLIENT/SERVER DEVELOPMENT

- Create Interface definition language (IDL) files
 - IDL files contain Globally unique identifier (GUID)
 - GUIDs must match: client and server compare GUIDs to verify proper versions of the distributed object
 - 128-bit binary number
- Next, add names of remote procs and params to IDL
- Then compile the IDL files
Compiler generates:
 - Header file (interface.h in C)
 - Client stub
 - Server stub

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L12.34

DCE - CLIENT-TO-SERVER BINDING

- Server name comes from directory server
- Server port comes from DCE daemon
 - DCE daemon has a well known port # client already knows

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L12.35

DCE - CLIENT TO SERVER BINDING - 2

- For a client to call a server, server must be registered
 - Java: uses RMI registry
- Client process to search for RMI server:
 - Locate the server's host machine
 - Locate the server (i.e. process) on the host
- Client must discover the server's RPC port
- DCE daemon:** maintains table of (server,port) pairs
- When servers boot:
 - Server asks OS for a port, registers port with DCE daemon
 - Also, server registers with directory server, separate server that tracks DCE servers

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L12.36

CH. 4.3: MESSAGE-ORIENTED COMMUNICATION

Apache ActiveMQ

L12.37

MESSAGE ORIENTED COMMUNICATION

- RPC assumes that the *client* and *server* are running **at the same time...** (*temporally coupled*)
- RPC communication is typically **synchronous**

- When client and server are not running at the same time
- Or when communications should not be **blocked...**

- Use case for **message-oriented communication**
 - Synchronous vs. asynchronous
 - Messaging systems
 - Message-queueing systems

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L12.38

SOCKETS

- Communication end point
- Applications can read / write data to
- Analogous to file streams for I/O, but **network streams**

| Operation | Description |
|-----------|---|
| socket | Create a new communication end point |
| bind | Attach local address to socket (IP / port) |
| listen | Tell OS what max # of pending connection requests should be |
| 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 |

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L12.39

SOCKETS - 2

- Servers execute 1st - 4 operations (socket, bind, listen, accept)
- Methods refer to C API functions
- Mappings across different libraries will vary (e.g. Java)

| Operation | Description |
|-----------|---|
| socket | Create a new communication end point |
| bind | Attach local address to socket (IP / port) |
| listen | Tell OS what max # of pending connection requests should be |
| 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 |

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L12.40

SERVER SOCKET OPERATIONS

- **Socket:** creates new communication end point
- **Bind:** associated IP and port with end point
- **Listen:** for connection-oriented communication, non-blocking call reserves buffers for specified number of pending connection requests server is willing to accept
- **Accept:** blocks until connection request arrives
 - Upon arrival, new socket is created matching original
 - Server spawns thread, or forks process to service incoming request
 - Server continues to wait for new connections on original socket

November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L12.41

CLIENT SOCKET OPERATIONS

- **Socket:** Creates socket client uses for communication
- **Connect:** Server transport-level address provided, client blocks until connection established
- **Send:** Supports sending data (to: server/client)
- **Receive:** Supports receiving data (from: server/client)
- **Close:** Closes communication channel
 - Analogous to closing a file stream


November 9, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L12.42

SOCKET COMMUNICATION

- Sockets provide primitives for implementing your own TCP/UDP communication protocols
- Directly using sockets for transient (non-persisted) messaging is very basic, can be brittle
 - Easy to make mistakes...
- Any extra communication facilities must be implemented by the application developer
- More advanced approaches are desirable
 - E.g. frameworks with support common desirable functionality


November 9, 2017 TCSS558: Applied Distributed Computing [Fall 2017]
Institute of Technology, University of Washington - Tacoma L12.43

QUESTIONS



November 9, 2017 TCSS558: Applied Distributed Computing [Fall 2017]
Institute of Technology, University of Washington - Tacoma L12.44

EXTRA SLIDES



45