Distributed Systems
Definitions

• Network
  – Communication paths for exchanging message among multiple system

• Distributed system
  – A collection of networked systems
  – Divide up a complex problem among systems
  – Access more resources
  – E.g., client/server distributed systems
    • Contact a web service such as Google or Facebook
    • Many machines cooperate to provide a service.
Challenges

• Failure
  ▪ Components of a distributed system fail.
    ➢ E.g., Programs, disks, networks
  ▪ Fault tolerance: entire system never fails despite the failure of components

• Performance
  ▪ A network connects a distributed system.
  ▪ Reduce the number of messages transferred
    ➢ Low latency, high bandwidth

• Security
  • Whether a remote web service is who you want to contact with
  • A third party does not monitor or change communication between you and a web service.
Communication Basics

• A distributed system does not have shared memory.
• Processes communicate via sending and receiving messages (packets).
• A router forwards packets from a sender to a receiver.

• Communication is not reliable. Packets may not reach destinations or get corrupted.
  ▪ Packet bits are flipped.
  ▪ Router or remote host are damaged.
  ▪ Lack of buffering (packet drops)
Communication Basics

• Protocol: Rules for defining the format and types of contents exchanged among systems

• Socket: A communication endpoint for reading and writing data

• Deal with packet loss
Unreliable Communication

• UDP (Universal Datagram Protocol)
  ▪ A receiver creates sockets.
  ▪ Another process sends datagrams to the receiver.
  ▪ Packets do not reach their destinations.

```
client code

int main(int argc, char* argv) {
int sd = UDP_open(20000);
struct sockaddr_in addrSnd, addrRcv;
int rc = UDP_FillSockAddr(&addrSnd, “machine.cs.wisc.edu”, 10000);
char message[BUFFER_SIZE[;
strcpy(message, “hello world”);
rc = UDP_Write(sd, &addrSnd, message, BUFFER_SIZE);
if(rc > 0) { int rc = UDP_Read(sd, &addrRcv, message, BUFFER_SIZE);} return 0;
}
```
int main(int argc, char* argv) {
    int sd = UDP_open(10000);
    assert(sd > -1);
    while (1) {
        struct sockaddr_in addr;
        char message[BUFFER_SIZE];
        rc = UDP_Read(sd, &addr, message, BUFFER_SIZE);
        if(rc > 0) {
            char reply[BUFFER_SIZE];
            sprintf(reply, "goodbye world");
            rc = UDP_Write(sd, &addr, reply, BUFFER_SIZE);
        }
    }
    return 0;
}
Reliable Communication

• Sender needs to know that receiver got a packet.

• Sender does not receive an ack.
  • Timeout/retry
  ▪ Sender sets a timer.
  ▪ If ack is not received within a timeout, sender retries sending the same message.
Reliable Communication

- Ack is lost.
- Sender re-sends a packet.
- Receiver receives the packet \textit{twice}.
  - A packet needs to be received exactly once.
  - E.g. packet content is “increment a counter”
Reliable Communication

• Solution
  ▪ Sender and receiver agree on a sequence counter.
  ▪ Sender labels a packet with the current value of the counter, e.g., N.
  ▪ Receiver increments the counter after receiving the message, e.g., N + 1.
  ▪ If ack is lost, the packet’s counter does not match the receiver’s counter.
    ▸ Receiver acks the packet but discards it.

• TCP (Transmission Control Protocol) is commonly used as a reliable communication.
Remote Procedure Call (RPC)

• Define an abstraction for communication

• Remote procedure call
  ▪ Server defines (exports) some routines.
  ▪ Client calls those routines locally.
  ▪ Results are returned to the client.

• Abstraction: called procedure runs on a local machine.

• Stub generator
  • A program that takes an interface and generates some pieces of code (stub)
  • **Client stub** contains functions.
  • Client program calls functions within the stub.
Remote Procedure Call (RPC)

• Client stub:
  ▪ Creates a message buffer
  ▪ Puts information into the buffer (marshaling or serialization)
    ➢ Function ID and input arguments
  ▪ Sends the message to server
  ▪ Waits for the reply
  ▪ Unpacks results (unmarshalling, deserialization)
  ▪ Returns to the client programs

• Server stub:
  ▪ Takes the information out of the message (unmarshalling or deserialization)
  ▪ Makes the actual call to the function
  ▪ Puts function results into a reply buffer (marshaling)
  ▪ Sends the reply to the client
Remote Procedure Call (RPC)

• Run-time library
  ▪ Finds the server that runs the desired service
  ▪ Use hostname and port number
  ▪ Choose a protocol for sending messages
    ▪ Message transfer over TCP has overhead.
      ➢ For each send-receive, two extra messages are sent (acks).
    ▪ Many RPC services are built on top of UDP.
      ➢ RPC provides the reliable communication.