

Recap

Network API: Socket

- Combination of IP, Port, and Protocol
- Server: bind, listen, accept
- Client: connect
- Client/Server: send/receive

Performance

Bandwidth

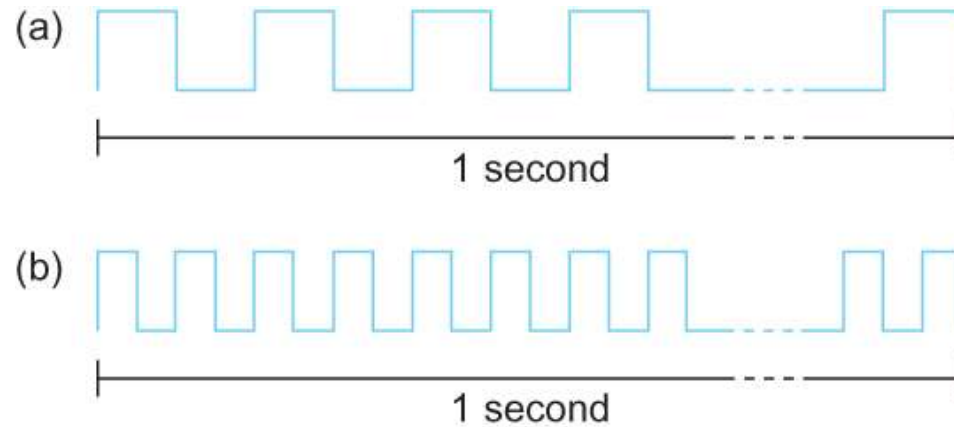
- Width of the frequency band
- **Number of bits per second** that can be transmitted over a communication link

1 Mbps: 1×10^6 bits/second = 1×2^{20} bits/sec

1×10^{-6} seconds to transmit each bit or imagine that a timeline:

- Each bit occupies 1 micro second space.
- On a 2 Mbps link the width is 0.5 micro second.
- Smaller the width more will be transmission per unit time.

Bandwidth



Bits transmitted at a particular bandwidth can be regarded as having some width:

- (a) bits transmitted at 1Mbps (each bit $1\ \mu\text{s}$ wide);
- (b) bits transmitted at 2Mbps (each bit $0.5\ \mu\text{s}$ wide).

Performance

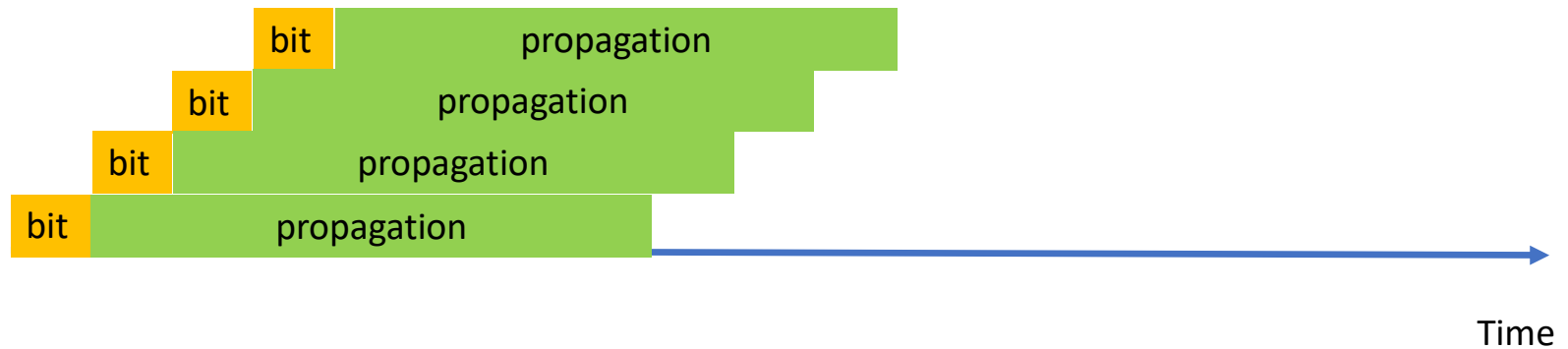
Latency = Transmit + Propagation + Queue

Propagation = distance/speed of light

Transmit = size/bandwidth

One bit transmission => propagation is important

Large bytes transmission => bandwidth (transmission time) is important



Delay vs Bandwidth

Relative importance of bandwidth and latency depends on application

- For large file transfer, bandwidth is critical

- For small messages (HTTP, NFS, etc.), latency is critical

- Variance in latency (jitter) can also affect some applications (e.g., audio/video conferencing)

Round Trip Time (RTT)

Consider that a sender often expects its receiver to reply after receiving.

- RTT: the delay for propagating a bit from the sender to the receiver and from the receiver to the sender.
- RTT/2: roughly the delay for propagating a bit from sender (or receiver) to receiver (or sender).

Example, in practice, the time for a packet to arrive at a receiver is:

$$\text{packetSize}/\text{bandwidth} + \text{RTT}/2$$

Let packet size = 1KB, bandwidth = 1Mbps, RTT = 50ms, how much time is needed for 1000 packet to arrive at the receiver?

$$(1000 * 1\text{KB} * 8\text{bits/Byte}) / 1\text{Mbps} + 50\text{ms}/2 = 8\text{s} + 25\text{ms} = 8.025\text{s}$$

Summary

We have identified what we expect from a computer network

We have defined a layered architecture for computer network that will serve as a blueprint for our design

We have discussed the socket interface which will be used by applications for invoking the services of the network subsystem

We have discussed two performance metrics using which we can analyze the performance of computer networks

Distributed Systems

(based on Ch. 48; focus on how to build systems for components that fail?)

Slides revised from
Matthew Tancreti
Iowa State University

Networks are Unreliable

Central tenet of networking: communication is unreliable

- Bits get flipped during transmission

- Routers may not work correctly

- Network cables get accidentally cut

- Packets arrive too quickly at a router and overfill its buffer space

- ...

Packet loss is fundamental in networking: how to deal with it?

Option 1: Make the Application Deal with Failure

One option is to not deal with packet loss

UDP/IP networking stack only provides application to send/receive individual packets (datagrams)

Provides no guarantee to application that packets will not be lost or arrive in different order than sent

Simple UDP Library Example

```
int UDP_Write(int sd, struct sockaddr_in *addr,
    char *buffer, int n) {
    int addr_len = sizeof(struct sockaddr_in);
    return sendto(sd, buffer, n, 0, (struct sockaddr *)
        addr, addr_len);
}

int UDP_Read(int sd, struct sockaddr_in *addr,
    char *buffer, int n) {
    int len = sizeof(struct sockaddr_in);
    return recvfrom(sd, buffer, n, 0, (struct sockaddr *)
        addr, (socklen_t *) &len);
}
```

← Send single packet

← Receive single packet

Figure 48.2: A Simple UDP Library (udp.c)

Option 2: Reliable Communication Layer

Second option is for transport layer to provide reliability

TCP/IP networking stack allows applications to send/receive full messages

The network stack breaks the message up into smaller packets, needs to deal with packet loss and out of order arrival of packets

Problem

How does sender know the receiver has received a packet?

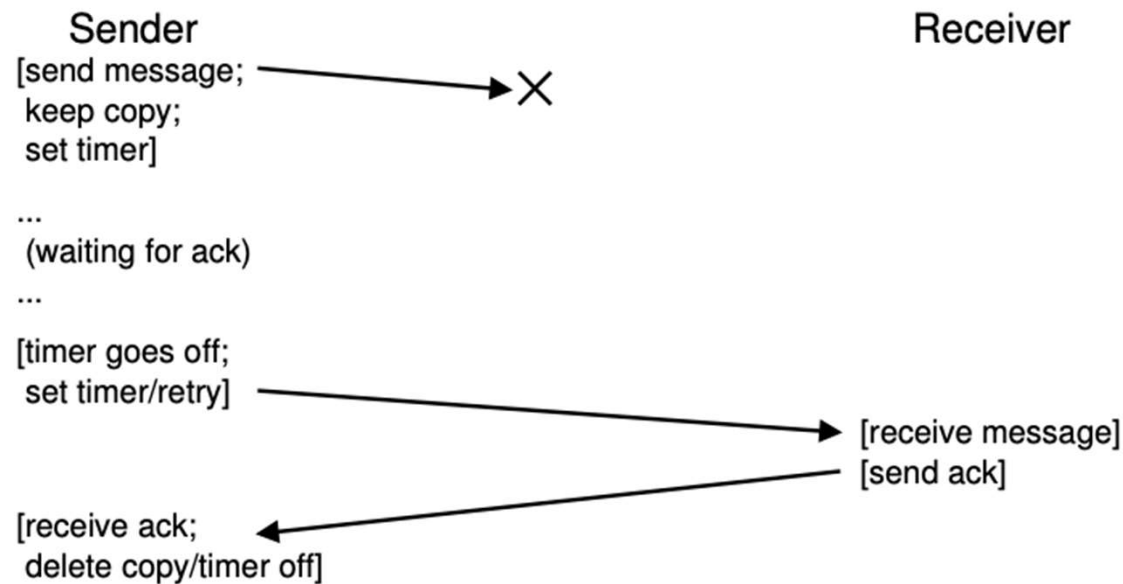
Does it wait for a message back from the receiver?

How long should it wait?

If packet is lost, how does receiver know sender tried to send a packet?

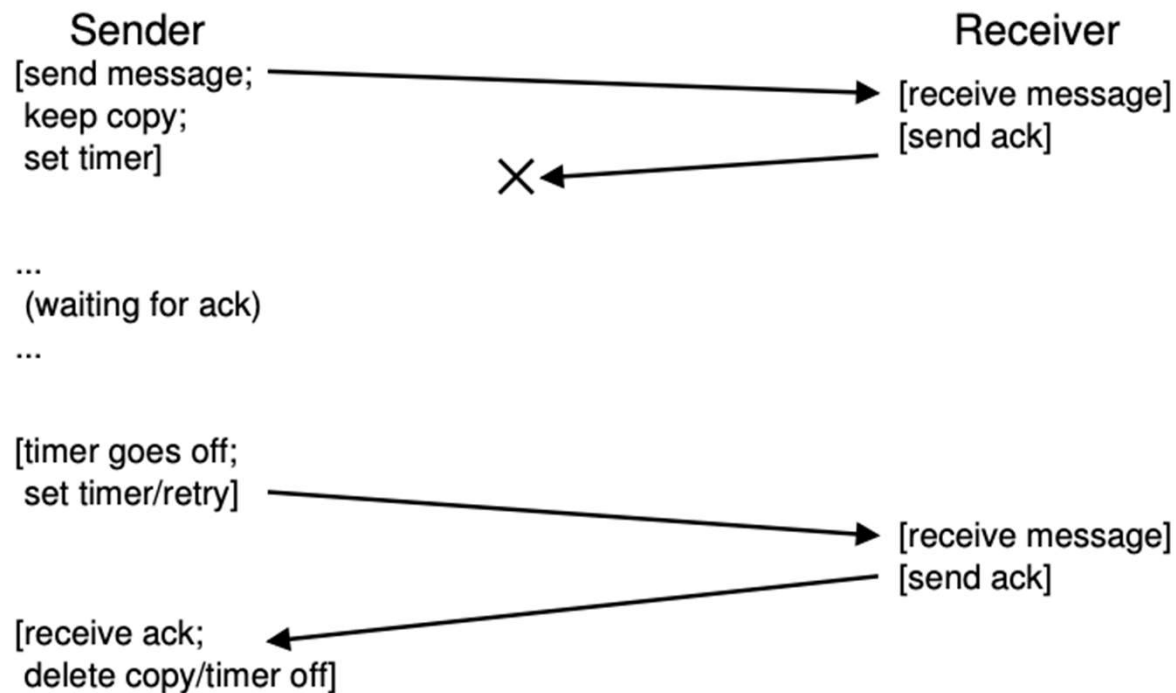
Acknowledgement with Timeout

Acknowledgement (ack) is technique where sender expects an acknowledgement within some amount of time, if there is a **timeout** the sender **retries** sending



Acknowledgement with Dropped Replay

Possible for receiver to get same packet multiple times



Sequence Number

Sender needs to know which packet was lost

Also want to prevent packets being added to messages multiple times

Sender attaches a **sequence number** (counter) to each packet

Receiver also keeps a count of received sequence numbers, sends sequence number with acknowledgement and checks to make sure packets are not provided to application twice

Remote Procedure Call

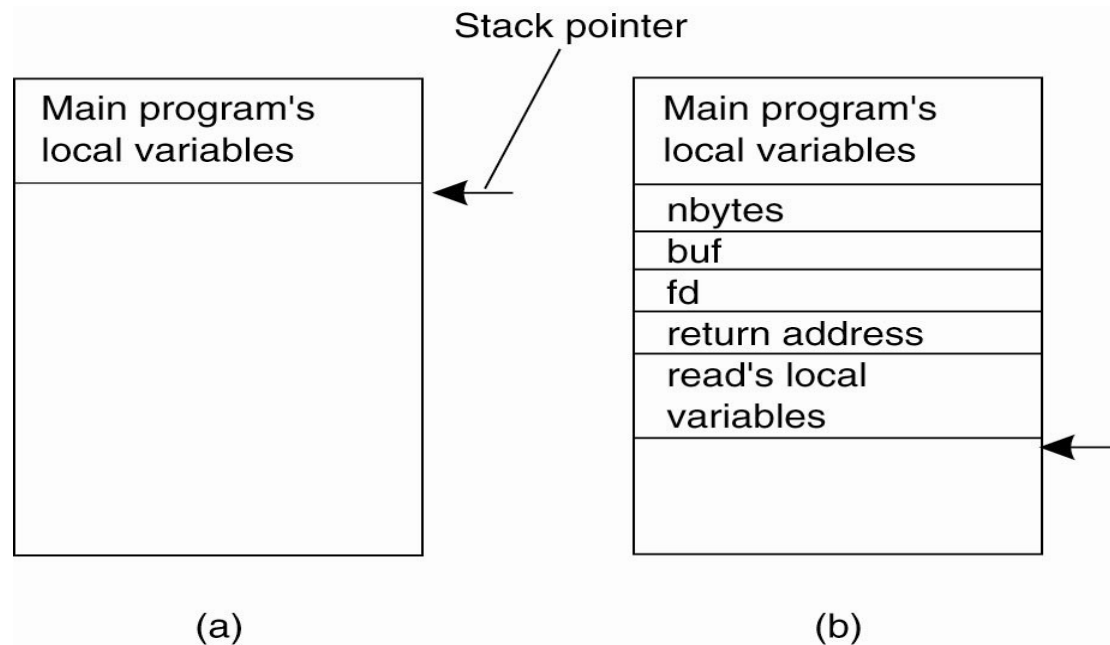
More than communication

Remote Procedure Call (RPC) has goal of making the process of executing code on a remote machine as simple and straightforward as calling a local function

A stub generator is used to remove the pain of packing function arguments and results into messages

Recall: Local Procedure Call

```
int main()
{
    ...
    read(fd,buf,nbytes);
    ...
}
```



(a) Parameter passing in a local procedure call: the stack before the call to `read`. (b) The stack while the called procedure is active.

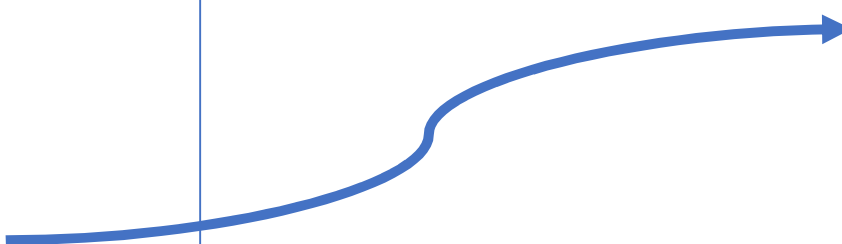
Remote Procedure Call: Example

Client

```
int main()
{
    ...
    k= add(i,j);
    ...
}
```

Server

```
int add(int i, int j)
{
    ...
    ...
}
```



RPC between Client and Server

RPC achieves transparency through client and server stubs

- **Client stub** packs parameters into a message to the server, and copies the result from the server to the client
- **Server stub** unpacks parameters, calls the procedure, and packs the result to the client

