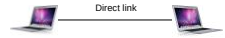
Distributed Systems

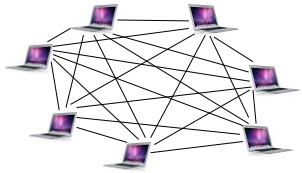
02. Networking

Paul Krzyzanowski
Rutgers University
Fall 2016

Connecting computers

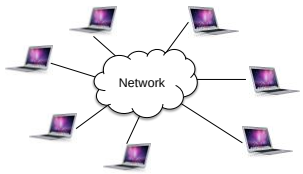
Point-to-point links
- Connect one sender with one receiver
- No conflict for access to link
- Not practical





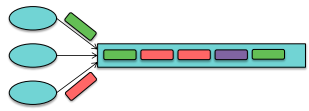
Connecting computers

Communication network
- Share the infrastructure
- Collision: when two nodes transmit at the same time, same channel
  - Both signals get damaged
- Multiple access problem
  - *How do you coordinate multiple senders?*



Multiple Access Protocols

- Share a communication medium
- Random access
  - Statistical multiplexing = **packet switching**
  - No timeslots
  - Anyone can transmit when ready
  - But be prepared for collisions or dropped packets



Modes of connection

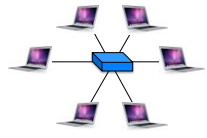
Circuit-switching (virtual circuit)
- Dedicated path (route) – established at setup
- Guaranteed (fixed) bandwidth – routers commit to resources
- Typically fixed-length packets (cells) – each cell only needs a virtual circuit ID
- Constant latency

This is what IP uses

Packet-switching (datagram)
- Shared connection; competition for use with others
- Data is broken into chunks called packets
- Each packet contains a destination address
- available bandwidth ≤ channel capacity
- variable latency

Ethernet

- Packet-based protocol
- Originally designed for shared (bus-based) links
- Each endpoint has a unique ethernet address
  - MAC address: 48-bit value



Ethernet service guarantees

- Each packet (frame) contains a CRC checksum
  - Recipient will drop the frame if it is bad
- No acknowledgement of packet delivery
- Unreliable, in-order delivery

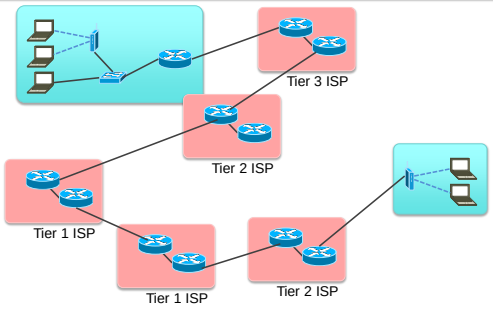
Going beyond the LAN

- LAN = Local Area Network
  - A set of devices connected to the same ethernet network is a LAN
  - Wi-Fi (802.11) is compatible with ethernet and is part of the LAN
- We want to communicate beyond the LAN
  - WAN = Wide Area Network
- The Internet
  - Evolved from ARPANET (1969)
  - Internet = global network of networks based on the Internet Protocol (IP) family of protocols

The Internet: Key Design Principles

1. Support interconnection of networks
   - No changes needed to the underlying physical network
   - IP is a *logical network*
2. Assume unreliable communication; design for best effort
   - If a packet does not get to the destination, software on the receiver will have to detect it and the sender will have to retransmit it
3. Routers connect networks
   - Store & forward delivery
   - They need not store information about the flow of packets
4. No global (centralized) control of the network

Routers tie LANs together into one Internet



A packet may pass through many networks – within and between ISPs

Protocols

What's in the data?

For effective communication
- same language, same conventions

For computers:
- electrical encoding of data
- where is the start of the packet?
- which bits contain the length?
- is there a checksum? where is it? how is it computed?
- what is the format of an address?
- byte ordering

These instructions and conventions are known as **protocols**

Layering

To ease software development and maximize flexibility:

– Network protocols are generally organized in layers

– Replace one layer without replacing surrounding layers

– Higher-level software does not have to know how to format an Ethernet packet
… or even know that Ethernet is being used

Layering

Most popular model of guiding (not specifying) protocol layers is

OSI reference model

Adopted and created by ISO

7 layers of protocols

OSI = Open Systems Interconnection
From the ISO = International Organization for Standardization

OSI Reference Model: Layer 1

Transmits and receives raw data to communication medium

Does not care about contents

Media, voltage levels, speed, connectors

Deals with representing bits

1 **Physical**

Examples: USB, Bluetooth, 1000BaseT, Wi-Fi

OSI Reference Model: Layer 2

Detects and corrects errors

Organizes data into frames before passing it down. Sequences packets (if necessary)

Accepts acknowledgements from immediate receiver

Deals with frames

2 **Data Link**
1 **Physical**

Examples: Ethernet MAC, PPP

OSI Reference Model: Layer 2

An ethernet switch is an example of a device that works on layer 2

It forwards ethernet frames from one host to another as long as the hosts are connected to the switch (switches may be cascaded)

This set of hosts and switches defines the local area network (LAN)

2 **Data Link**
1 **Physical**

OSI Reference Model: Layer 3

Relay and route information to destination

Manage journey of datagrams and figure out intermediate hops (if needed)

3 **Network**
2 **Data Link**
1 **Physical**

Deals with datagrams

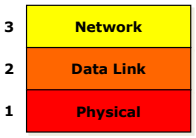
Examples: IP, X.25

OSI Reference Model: Layer 3

An IP router is an example of a device that works on layer 3

A router takes an incoming IP packet and determines which interface to send it out

It enables multiple networks to be connected together

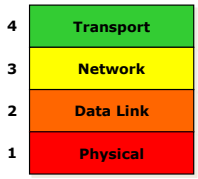


Cisco CRS 4-Slot Single Shelf System

OSI Reference Model: Layer 4

Provides an interface for end-to-end (application-to-application) communication: sends & receives segments of data. Manages flow control. May include end-to-end reliability

Network interface is similar to a mailbox



Deals with segments

Examples: TCP, UDP

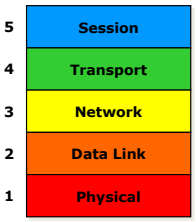
OSI Reference Model: Layer 5

Services to coordinate dialogue and manage data exchange

Software implemented switch

Manage multiple logical connections

Keep track of who is talking: establish & end communications



Deals with data streams

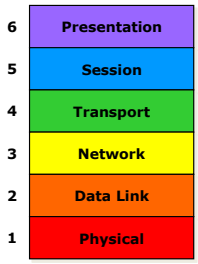
Examples: HTTP 1.1, SSL

OSI Reference Model: Layer 6

Data representation

Concerned with the meaning of data bits

Convert between machine representations

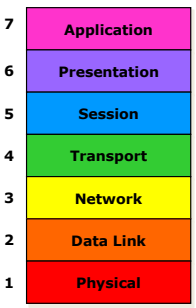


Deals with objects

Examples: XDR, ASN.1, MIME, JSON, XML

OSI Reference Model: Layer 7

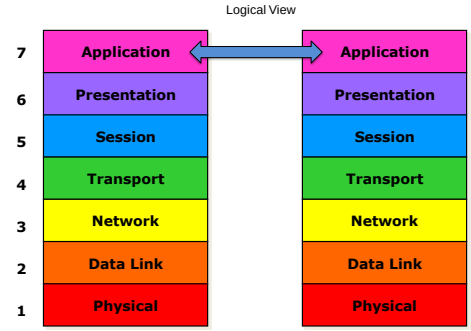
Collection of application-specific protocols



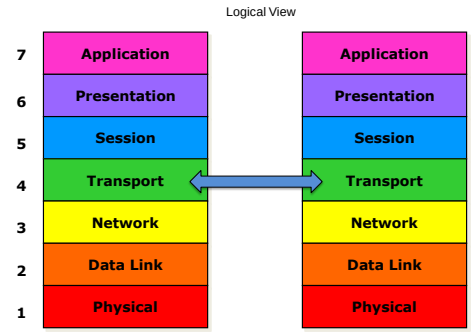
Deals with app-specific protocols

Examples:
web (HTTP)
email (SMTP, POP, IMAP)
file transfer (FTP)
directory services (LDAP)

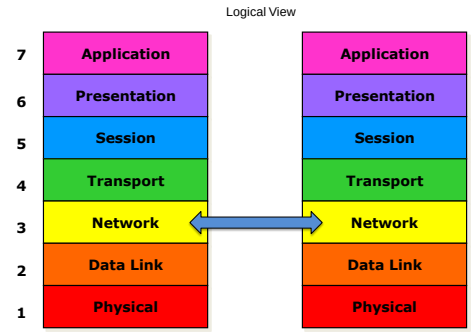
A layer communicates with its counterpart



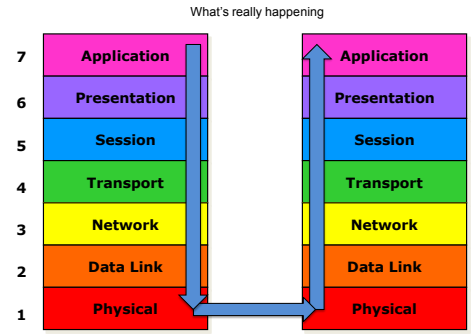
A layer communicates with its counterpart



A layer communicates with its counterpart



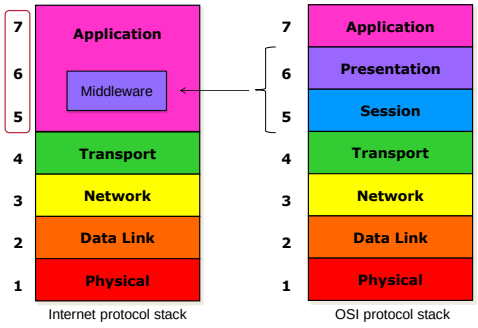
But really traverses the stack



Internet Protocol

- A set of protocols designed to handle the interconnection of a large number of local and wide-area networks that comprise the Internet
- IPv4 & IPv6: network layer
  - Other protocols include TCP, UDP, RSVP, ICMP, etc.
  - Relies on routing from one physical network to another
  - IP is connectionless
    - No state needs to be saved at each router
  - Survivable design: support multiple paths for data
    - … but packet delivery is not guaranteed!

IP vs. OSI stack



Protocol Encapsulation

At any layer
- The higher level protocol headers are just treated like data
- Lower level protocol headers can be ignored

An ethernet switch or ethernet driver sees this:

A router or IP driver sees this:

A TCP driver sees this:

An application sees this:

Client-Server Communication

Addressing machines (data link layer)

Each interface on a host has a unique MAC address
- E.g., aramis.rutgers.edu: 48-bit ethernet address = = 00:03:ba:09:1b:b0

- This isn't too interesting to us as programmers
  - We usually don't communicate at the data link layer

Addressing machines (network layer)

Each interface on a host is given a unique IP address
- IPv4 (still the most common in the U.S.): 32-bit number
  - Example, cs.rutgers.edu = 128.6.4.2 = 0x80060402
- IPv6: 128-bit number
  - Example, cs.rutgers.edu = 0:0:0:0:0:FFFF:128.6.4.2 = = ::FFFF:8006:0402

This is a mixed hex/decimal notation to embed IPv4 addresses

But we want to talk with applications … not just the hosts

Addressing applications (transport layer)

Communication endpoint at the machine
- **Port number**: 16-bit number
- Port number = transport endpoint
  - Identifies a specific data stream
- Some services use well-known port numbers (0 – 1023)
  - IANA: Internet Assigned Numbers Authority (www.iana.org)
  - Also see the file /etc/services
    ftp: 21/TCP ssh: 22/tcp smtp: 25/tcp http: 80/tcp ntp: 123/udp
- Ports for proprietary apps: 1024 – 49151
- Dynamic/private ports: 49152 – 65535

- To communicate with applications, we use a transport layer protocol and an IP address and port number

IP transport layer protocols

IP gives us two transport-layer protocols for communication

- **TCP: Transmission Control Protocol**
  - Connection-oriented service – operating system keeps state
  - Full-duplex connection: both sides can send messages over the same link
  - Reliable data transfer: the protocol handles retransmission
  - In-order data transfer: the protocol keeps track of sequence numbers
  - Flow control: receiver stops sender from sending too much data
  - Congestion control: "plays nice" on the network – reduce transmission rate
  - 20-byte header
- **UDP: User Datagram Protocol**
  - Connectionless service: lightweight transport layer over IP
  - Data may be lost
  - Data may arrive out of sequence
  - Checksum for corrupt data: operating system drops bad packets
  - 8-byte header

Network API

- App developers need access to the network
- A *Network Application Programming Interface (API)* provides this
  - Core services provided by the operating system
    - Operating System controls access to resources
  - Libraries may handle the rest

Programming: connection-oriented protocols

		analogous to phone call
1.	establish connection	*dial phone number*
2.	[negotiate protocol]	*[decide on a language]*
3.	exchange data	*speak*
4.	terminate connection	*hang up*

Reliable byte stream service

- provides illusion of having a dedicated circuit
- messages guaranteed to arrive in-order
- application does not have to address each message

Programming: connectionless protocols

analogous to mailbox

- no call setup
- send/receive data (each packet addressed) — *drop letter in mailbox (each letter addressed)*
- no termination

Datagram service

- client is not positive whether message arrived at destination
- no state has to be maintained at client or server
- cheaper but less reliable than virtual circuit service

Sockets

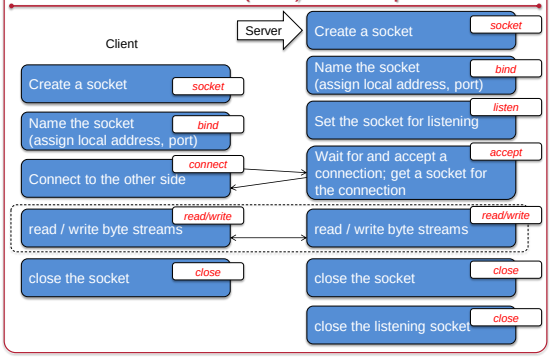
- Dominant API for transport layer connectivity
- Created at UC Berkeley for 4.2BSD Unix (1983)
- Design goals
  - Communication between processes should not depend on whether they are on the same machine
  - Communication should be efficient
  - Interface should be compatible with files
  - Support different protocols and naming conventions
    - *Sockets is not just for the Internet Protocol family*

What is a socket?

Abstract object from which messages are sent and received

- Looks like a file descriptor
- Application can select particular style of communication
  - Virtual circuit (connection-oriented), datagram (connectionless), message-based, in-order delivery
- Unrelated processes should be able to locate communication endpoints
  - Sockets can have a name
  - Name should be meaningful in the communications domain
    - E.g., Address & port for IP communications

Connection-Oriented (TCP) socket operations



Python Example

Note: try/except blocks are missing

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
remote_addr = socket.gethostbyname(host)
s.connect(remote_addr, port)
s.sendall(message)
# …
```

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(5)

while 1:
    conn, addr = s.accept()
    # do work on socket conn
    msg = conn.recv()
s.close
```

Java provides shortcuts that combine calls

Example

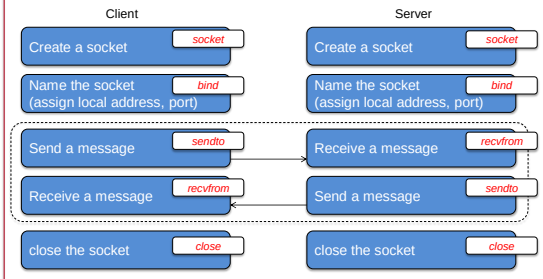
Java

```
Socket s = new Socket("www.rutgers.edu", 2211)
```

C

```
int s = socket(AF_INET, SOCK_STREAM, 0);

struct sockaddr_in myaddr; /* initialize address structure */
myaddr.sin_family = AF_INET;
myaddr.sin_addr.s_addr = htonl(INADDR_ANY);
myaddr.sin_port = htons(0);

bind(s, (struct sockaddr *)&myaddr, sizeof(myaddr));

/* look up the server's address */
struct hostent *hp;      /* host information */
struct sockaddr_in servaddr;    /* server address */

memset((char*)&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(2211);
hp = gethostbyname("www.rutgers.edu");

if (connect(fd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        /* connect failed */
}
```


Connectionless (UDP) socket operations



The end