Distributed Systems

02. Networking

Paul Krzyzanowski
Rutgers University
Fall 2018
Inter-computer communication

- Without shared memory, computers need to communicate

Direct links aren't practical – they don't scale
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?*
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

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 \( \leq \) channel capacity
- Variable latency

This is what IP uses
Packet switching

- Random access
  - Statistical multiplexing
  - No timeslots
  - Anyone can transmit when ready
  - But be prepared for collisions or dropped packets
Ethernet

- Packet-based protocol
- Originally designed for shared (bus-based) links
- Each endpoint has a unique ethernet address
  - MAC address: 48-bit number
Local Area Network: **Data Link Layer**

**Hub:**
- Device that acts as a central point for LAN cables
- Take incoming data from one port & send to all other ports

**Switch**
- Moves data from input to output port
- Analyzes packet to determine destination port and makes a virtual connection between the ports
- Scales better than a hub

Link-layer switches: create a physical network (e.g., Ethernet, Wi-Fi)
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
  – Packet loss possible
Going beyond the LAN

• We want to communicate beyond the LAN
  – WAN = Wide Area Network

• Network Layer
  – Responsible for routing between LANs

• The Internet
  – Evolved from ARPANET (1969)
  – Internet = global network of networks based on the Internet Protocol (IP) family of protocols
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 IP-based 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!
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
   - 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

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
IP addressing

• Each network endpoint has a unique IP address
  – No relation to an ethernet address
  – IPv4: 32-bit address
  – IPv6: 128-bit address

• Data is broken into packets
  – Each packet contains source & destination IP addresses

• IP gives us machine-to-machine communication
Transport Layer: UDP & TCP
Transport Layer

• We want to communicate between applications

• The transport layer gives us logical "channels" for communication
  – Processes can write to and receive from these channels

• Two transport layer protocols in IP are TCP & UDP
  – A port number identifies a unique channel on each computer
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
Reliable delivery

A

msg 1

B

A

ack 1

B

A

msg 2

B

A

ack 2

B

A

msg 3

B

A

ack 3

B
Reliable delivery

- This slows us down A LOT!
  - Cannot send a message until the previous one reaches the destination AND the acknowledgement comes back
Transmit up to $N$ messages

• Pipelining
  – Send a bunch of packets without waiting for an ack from each one

• Piggybacked acknowledgements
  – Don't waste a separate acknowledgement message
  – If we have data to send back, send the ack in that packet

• Cumulative acknowledgements
  – If we have no data, don't send lots of individual acks
  – Cumulative ack = "the next byte I need" – byte count of all bytes received so far

• TCP uses all of these
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

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

Examples: Ethernet MAC, PPP
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)**.
OSI Reference Model: Layer 3

Relay and route information to destination

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

Examples: IP, X.25
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.

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

Examples: HTTP 1.1, SSL
OSI Reference Model: Layer 6

Data representation
Concerned with the meaning of data bits
Convert between machine representations

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

Deals with objects
OSI Reference Model: Layer 7

Collection of application-specific protocols

Examples:
- web (HTTP)
- email (SMTP, POP, IMAP)
- file transfer (FTP)
- directory services (LDAP)
A layer communicates with its counterpart
IP vs. OSI stack

Internet protocol stack

OSI protocol stack

Application

Transport (TCP, UDP)

Network (IP)

Data Link

Physical

Middleware

Presentation

Session

Transport

Network

Data Link

Physical

September 10, 2018

© 2014-2018 Paul Krzyzanowski
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:
Programming for networking
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

1. establish connection
2. [negotiate protocol]
3. exchange data
4. terminate connection

Reliable byte stream service (TCP)

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

analogous to phone call

1. dial phone number
2. [decide on a language]
3. speak
4. hang up
Programming: connectionless protocols

- no call setup
- send/receive data
  (each packet addressed)
- no termination

Datagram service (UDP)

- client is not positive whether message arrived at destination
- no state has to be maintained at client or server

analogous to mailbox

drop letter in mailbox
  (each letter addressed)
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

Client

1. Create a socket
2. Name the socket (assign local address, port)
3. Connect to the other side
4. read / write byte streams
5. close the socket

Server

1. Create a socket
2. Name the socket (assign local address, port)
3. Set the socket for listening
4. Wait for and accept a connection; get a socket for the connection
5. read / write byte streams
6. close the socket
7. close the listening socket

September 10, 2018
© 2014-2018 Paul Krzyzanowski
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 */
}
```
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
Connectionless (UDP) socket operations

Client

- Create a socket
- Name the socket (assign local address, port)
- Send a message
- Receive a message
- close the socket

Server

- Create a socket
- Name the socket (assign local address, port)
- Receive a message
- Send a message
- close the socket
The end