Network Programming: Part I

15-213 / 18-213: Introduction to Computer Systems
20th Lecture, Nov. 4, 2014

Instructors:
Greg Ganger, Greg Kesden, and Dave O’Hallaron
A Client-Server Transaction

1. Client sends request
2. Server handles request
3. Server sends response
4. Client handles response

Note: clients and servers are processes running on hosts (can be the same or different hosts)

Most network applications are based on the client-server model:

- A server process and one or more client processes
- Server manages some resource
- Server provides service by manipulating resource for clients
- Server activated by request from client (vending machine analogy)
Hardware Organization of a Network Host

- CPU chip
- ALU
- register file
- MI

- System bus
- Memory bus

- Main memory
- I/O bus

- Expansion slots

- USB controller
- Graphics adapter
- Disk controller
- Network adapter

- Mouse
- Keyboard
- Monitor
- Disk
- Network
Computer Networks

- A **network** is a hierarchical system of boxes and wires organized by geographical proximity
  - SAN (System Area Network) spans cluster or machine room
    - Switched Ethernet, Quadrics QSW, ...
  - LAN (Local Area Network) spans a building or campus
    - Ethernet is most prominent example
  - WAN (Wide Area Network) spans country or world
    - Typically high-speed point-to-point phone lines

- An **internetwork (internet)** is an interconnected set of networks
  - The Global IP Internet (uppercase “I”) is the most famous example of an internet (lowercase “i”)

- Let’s see how an internet is built from the ground up
Lowest Level: Ethernet Segment

- Ethernet segment consists of a collection of *hosts* connected by wires (twisted pairs) to a *hub*
- Spans room or floor in a building
- **Operation**
  - Each Ethernet adapter has a unique 48-bit address (MAC address)
    - E.g., 00:16:ea:e3:54:e6
  - Hosts send bits to any other host in chunks called *frames*
  - Hub slavishly copies each bit from each port to every other port
    - Every host sees every bit
    - Note: Hubs are on their way out. Bridges (switches, routers) became cheap enough to replace them
Next Level: Bridged Ethernet Segment

- Spans building or campus
- Bridges cleverly learn which hosts are reachable from which ports and then selectively copy frames from port to port
Conceptual View of LANs

For simplicity, hubs, bridges, and wires are often shown as a collection of hosts attached to a single wire:
Next Level: internets

- Multiple incompatible LANs can be physically connected by specialized computers called *routers*.
- The connected networks are called an *internet* (lower case).

LAN 1 and LAN 2 might be completely different, totally incompatible (e.g., Ethernet, Fibre Channel, 802.11*, T1-links, DSL, ...)

![Diagram](image-url)
Logical Structure of an internet

- **Ad hoc interconnection of networks**
  - No particular topology
  - Vastly different router & link capacities

- **Send packets from source to destination by hopping through networks**
  - Router forms bridge from one network to another
  - Different packets may take different routes
The Notion of an internet Protocol

- How is it possible to send bits across incompatible LANs and WANs?

- Solution: *protocol* software running on each host and router
  - Protocol is a set of rules that governs how hosts and routers should cooperate when they transfer data from network to network.
  - Smooths out the differences between the different networks
What Does an internet Protocol Do?

- Provides a naming scheme
  - An internet protocol defines a uniform format for *host addresses*
  - Each host (and router) is assigned at least one of these internet addresses that uniquely identifies it

- Provides a delivery mechanism
  - An internet protocol defines a standard transfer unit (*packet*)
  - Packet consists of *header* and *payload*
    - Header: contains info such as packet size, source and destination addresses
    - Payload: contains data bits sent from source host
Transferring internet Data Via Encapsulation

LAN1

Host A
client

protocol software

LAN1 adapter

(1) data

internet packet

LAN1 frame

(2) data PH FH1

LAN1 adapter

(3) data PH FH1

Host B
server

protocol software

LAN2 adapter

(4) data PH FH1

(5) data PH FH2

Router

LAN1 adapter

LAN2 adapter

LAN2 frame

(6) data PH FH2

(7) data PH FH2

(8) data

LAN2

PH: Internet packet header
FH: LAN frame header
Other Issues

- We are glossing over a number of important questions:
  - What if different networks have different maximum frame sizes? (segmentation)
  - How do routers know where to forward frames?
  - How are routers informed when the network topology changes?
  - What if packets get lost?

- These (and other) questions are addressed by the area of systems known as computer networking
Global IP Internet (upper case)

- Most famous example of an internet

- Based on the TCP/IP protocol family
  - IP (Internet Protocol):
    - Provides *basic naming scheme* and unreliable *delivery capability* of packets (datagrams) from *host-to-host*
  - UDP (Unreliable Datagram Protocol)
    - Uses IP to provide *unreliable* datagram delivery from *process-to-process*
  - TCP (Transmission Control Protocol)
    - Uses IP to provide *reliable* byte streams from *process-to-process* over *connections*

- Accessed via a mix of Unix file I/O and functions from the *sockets interface*
Hardware and Software Organization of an Internet Application

**Internet client host**
- **Client**: User code
- **TCP/IP**: Kernel code
- **Network adapter**: Hardware and firmware

**Internet server host**
- **Server**: TCP/IP
- **Network adapter**: 

**Global IP Internet**

*Hardware interface (interrupts)*
*Sockets interface (system calls)*
A Programmer’s View of the Internet

1. Hosts are mapped to a set of 32-bit IP addresses
   - 128.2.203.179

2. The set of IP addresses is mapped to a set of identifiers called Internet domain names
   - 128.2.203.179 is mapped to www.cs.cmu.edu

3. A process on one Internet host can communicate with a process on another Internet host over a connection
Aside: IPv4 and IPv6

- The original Internet Protocol, with its 32-bit addresses, is known as Internet Protocol Version 4 (IPv4)

- 1996: Internet Engineering Task Force (IETF) introduced Internet Protocol Version 6 (IPv6) with 128-bit addresses
  - Intended as the successor to IPv4

- As of 2014, vast majority of Internet traffic still carried by IPv4
  - Only 4% of users access Google services using IPv6.

- We will focus on IPv4, but will show you how to write networking code that is protocol-independent.
  - Not covered in your textbook
(1) IP Addresses

- 32-bit IP addresses are stored in an **IP address struct**
  - IP addresses are always stored in memory in **network byte order** (big-endian byte order)
  - True in general for any integer transferred in a packet header from one machine to another.
    - E.g., the port number used to identify an Internet connection.

```c
/* Internet address structure */
struct in_addr {
    uint32_t s_addr; /* network byte order (big-endian) */
};
```

**Useful network byte-order conversion functions ("l" = 32 bits, "s" = 16 bits)**

- `htonl`: convert `uint32_t` from host to network byte order
- `htons`: convert `uint16_t` from host to network byte order
- `ntohl`: convert `uint32_t` from network to host byte order
- `ntohs`: convert `uint16_t` from network to host byte order
Dotted Decimal Notation

- By convention, each byte in a 32-bit IP address is represented by its decimal value and separated by a period
  - IP address: $0x8002C2F2 = 128.2.194.242$

- Functions for converting between binary IP addresses and dotted decimal strings:
  - `inet_pton`: dotted decimal string $\rightarrow$ IP address in network byte order
  - `inet_ntop`: IP address in network byte order $\rightarrow$ dotted decimal string

- “n” denotes network
- “p” denotes presentation
(2) Internet Domain Names

```
unnamed root

.net  .edu  .gov  .com

mit  cmu  berkeley  amazon

.cs  .ece  .www

ics  pdl

whaleshark  www
128.2.210.175  128.2.131.66
```

First-level domain names

Second-level domain names

Third-level domain names
Domain Naming System (DNS)

- The Internet maintains a mapping between IP addresses and domain names in a huge worldwide distributed database called DNS.

- Conceptually, programmers can view the DNS database as a collection of millions of host entries.
  - Each host entry defines the mapping between a set of domain names and IP addresses.
  - In a mathematical sense, a host entry is an equivalence class of domain names and IP addresses.
Properties of DNS Mappings

- Can explore properties of DNS mappings using `nslookup`
  - Output edited for brevity

- Each host has a locally defined domain name `localhost` which always maps to the *loopback address* `127.0.0.1`

  ```
  linux> nslookup localhost
  Address: 127.0.0.1
  ```

- Use `hostname` to determine real domain name of local host:

  ```
  linux> hostname
  whaleshark.ics.cs.cmu.edu
  ```
Properties of DNS Mappings (cont)

- Simple case: one-to-one mapping between domain name and IP address:

  ```
  linux> nslookup whaleshark.ics.cs.cmu.edu
  Address: 128.2.210.175
  ```

- Multiple domain names mapped to the same IP address:

  ```
  linux> nslookup cs.mit.edu
  Address: 18.62.1.6
  linux> nslookup eecs.mit.edu
  Address: 18.62.1.6
  ```
Properties of DNS Mappings (cont)

- Multiple domain names mapped to multiple IP addresses:

  ```
  linux> nslookup www.twitter.com
  Address: 199.16.156.6
  Address: 199.16.156.70
  Address: 199.16.156.102
  Address: 199.16.156.230
  ```

  ```
  linux> nslookup twitter.com
  Address: 199.16.156.102
  Address: 199.16.156.230
  Address: 199.16.156.6
  Address: 199.16.156.70
  ```

- Some valid domain names don’t map to any IP address:

  ```
  linux> nslookup ics.cs.cmu.edu
  *** Can't find ics.cs.cmu.edu: No answer
  ```
(3) Internet Connections

- Clients and servers communicate by sending streams of bytes over *connections*. Each connection is:
  - *Point-to-point*: connects a pair of processes.
  - *Full-duplex*: data can flow in both directions at the same time,
  - *Reliable*: stream of bytes sent by the source is eventually received by the destination in the same order it was sent.

- A *socket* is an endpoint of a connection
  - *Socket address* is an `IPaddress:port` pair

- A *port* is a 16-bit integer that identifies a process:
  - *Ephemeral port*: Assigned automatically by client kernel when client makes a connection request.
  - *Well-known port*: Associated with some *service* provided by a server (e.g., port 80 is associated with Web servers)
Well-known Ports and Service Names

- Popular services have permanently assigned *well-known ports and corresponding well-known service names*:
  - echo server: 7/echo
  - ssh servers: 22/ssh
  - email server: 25/smtp
  - web servers: 80/http

- Mappings between well-known ports and service names is contained in the file `/etc/services` on each Linux machine.
Anatomy of a Connection

- A connection is uniquely identified by the socket addresses of its endpoints (*socket pair*)
  - (cliaddr:cliport, servaddr:servport)

**Client socket address**

- 128.2.194.242:51213

**Server socket address**

- 208.216.181.15:80

**Client host address**

- 128.2.194.242

**Server host address**

- 208.216.181.15

51213 is an ephemeral port allocated by the kernel

80 is a well-known port associated with Web servers
Using Ports to Identify Services

Client host

Service request for 128.2.194.242:80 (i.e., the Web server)

Server host 128.2.194.242

Web server (port 80)

Echo server (port 7)

Client host

Service request for 128.2.194.242:7 (i.e., the echo server)

Web server (port 80)

Echo server (port 7)
Sockets Interface

- Set of system-level functions used in conjunction with Unix I/O to build network applications.

- Created in the early 80’s as part of the original Berkeley distribution of Unix that contained an early version of the Internet protocols.

- Available on all modern systems
  - Unix variants, Windows, OS X, IOS, Android, ARM
Sockets

- **What is a socket?**
  - To the kernel, a socket is an endpoint of communication
  - To an application, a socket is a file descriptor that lets the application read/write from/to the network
    - *Remember:* All Unix I/O devices, including networks, are modeled as files

- Clients and servers communicate with each other by reading from and writing to socket descriptors

- The main distinction between regular file I/O and socket I/O is how the application “opens” the socket descriptors
Socket Address Structures

- **Generic socket address:**
  - For address arguments to `connect`, `bind`, and `accept`
  - Necessary only because C did not have generic (void *) pointers when the sockets interface was designed
  - For casting convenience, we adopt the Stevens convention:
    ```c
    typedef struct sockaddr SA;
    
    struct sockaddr {
        uint16_t sa_family; /* Protocol family */
        char sa_data[14]; /* Address data. */
    };
    ```

| sa_family | Family Specific |
Socket Address Structures

- Internet-specific socket address:
  - Must cast `struct sockaddr_in *` to `(SA *)` for functions that take socket address arguments.

```c
struct sockaddr_in  {
    uint16_t sin_family;  /* Protocol family (always AF_INET) */
    uint16_t sin_port;    /* Port num in network byte order */
    struct in_addr sin_addr;  /* IP addr in network byte order */
    unsigned char sin_zero[8];/* Pad to sizeof(struct sockaddr) */
};
```

<table>
<thead>
<tr>
<th>sa_family</th>
<th>sin_port</th>
<th>sin_addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>AF_INET</td>
<td></td>
<td>0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Family Specific
Sockets Interface: `socket`

- Clients and servers use the `socket` function to create a `socket descriptor`:

  ```c
  int socket(int domain, int type, int protocol)
  ```

- **Example:**

  ```c
  int clientfd = Socket(AF_INET, SOCK_STREAM, 0);
  ```

  Indicates that we are using 32-bit IPV4 addresses

  Indicates that the socket will be the end point of a connection

Protocol specific! Best practice is to use `getaddrinfo` to generate the parameters automatically, so that code is protocol independent.
Sockets Interface

Client

- `getaddrinfo`
- `socket`
- `connect`
- `rio_readlineb`
- `rio_writen`
- `close`

Server

- `getaddrinfo`
- `socket`
- `bind`
- `listen`
- `accept`
- `rio_readlineb`
- `rio_writen`
- `close`

Client / Server Session

- `open_clientfd`
- `open_listenfd`

Connection request

Await connection request from next client
Sockets Interface: connect

- A client establishes a connection with a server by calling `connect`:
  
  ```c
  int connect(int clientfd, SA *addr, socklen_t addrlen);
  ```

- Attempts to establish a connection with server at socket address `addr`
  
  - If successful, then `clientfd` is now ready for reading and writing.
  - Resulting connection is characterized by socket pair
    
    `x:y, addr.sin_addr:addr.sin_port`
    
    - `x` is client address
    - `y` is ephemeral port that uniquely identifies client process on client host

Best practice is to use `getaddrinfo` to supply the arguments `addr` and `addrlen`. 
Sockets Interface

Client
- `getaddrinfo`
- `socket`
- `connect`
- `rio_readlineb`
- `rio_writen`
- `close`

Server
- `getaddrinfo`
- `socket`
- `bind`
- `listen`
- `accept`
- `rio_readlineb`
- `rio_writen`
- `close`

Connection request

Await connection request from next client

open_clientfd

open_listenfd

Client / Server Session

EOF

37
Sockets Interface: bind

- A server uses `bind` to ask the kernel to associate the server’s socket address with a socket descriptor:

  ```c
  int bind(int sockfd, SA *addr, socklen_t addrlen);
  ```

- The process can read bytes that arrive on the connection whose endpoint is `addr` by reading from descriptor `sockfd`.

- Similarly, writes to `sockfd` are transferred along connection whose endpoint is `addr`.

Best practice is to use `getaddrinfo` to supply the arguments `addr` and `addrlen`.
Sockets Interface

Client

Client / Server Session

open_clientfd

Server

open_listenfd

getaddrinfo

socket

connect

rio_readlineb

rio_writen

close

getaddrinfo

socket

bind

listen

accept

rio_readlineb

rio_writen

close

Await connection request from next client

Connection request

EOF
Sockets Interface: `listen`

- By default, kernel assumes that descriptor from socket function is an *active socket* that will be on the client end of a connection.

- A server calls the `listen` function to tell the kernel that a descriptor will be used by a server rather than a client:

  ```c
  int listen(int sockfd, int backlog);
  ```

- Converts `sockfd` from an active socket to a *listening socket* that can accept connection requests from clients.

- `backlog` is a hint about the number of outstanding connection requests that the kernel should queue up before starting to refuse requests.
Sockets Interface

Client
- getaddrinfo
- socket
- connect
- rio_readlineb
- rio_writen
- close

Server
- getaddrinfo
- socket
- bind
- listen
- accept

open_clientfd

Connection request

Await connection request from next client

open_listenfd

Client / Server Session

Sockets Interface: accept

- Servers wait for connection requests from clients by calling `accept`:

  ```c
  int accept(int listenfd, SA *addr, int *addrlen);
  ```

- Waits for connection request to arrive on the connection bound to `listenfd`, then fills in client’s socket address in `addr` and `sizeof` socket address in `addrlen`.

- Returns a `connected descriptor` that can be used to communicate with the client via Unix I/O routines.
accept Illustrated

1. Server blocks in accept, waiting for connection request on listening descriptor `listenfd`

2. Client makes connection request by calling and blocking in `connect`

3. Server returns `connfd` from accept. Client returns from `connect`. Connection is now established between `clientfd` and `connfd`
Connected vs. Listening Descriptors

- **Listening descriptor**
  - End point for client connection requests
  - Created once and exists for lifetime of the server

- **Connected descriptor**
  - End point of the connection between client and server
  - A new descriptor is created each time the server accepts a connection request from a client
  - Exists only as long as it takes to service client

- **Why the distinction?**
  - Allows for concurrent servers that can communicate over many client connections simultaneously
    - E.g., Each time we receive a new request, we fork a child to handle the request
Sockets Interface

Client

- getaddrinfo
- socket
- connect
- rio_readlineb
- rio_writen
- close

Server

- getaddrinfo
- socket
- bind
- listen
- accept
- rio_readlineb
- rio_writen
- close

Connection request

open_clientfd

open_listenfd

Await connection request from next client

Client / Server Session
Next time

- Using `getaddrinfo` for host and service conversion
- Writing clients and servers
- Writing Web servers!
Additional slides
Basic Internet Components

- Internet backbone:
  - collection of routers (nationwide or worldwide) connected by high-speed point-to-point networks

- Internet Exchange Points (IXP):
  - router that connects multiple backbones (often referred to as peers)
  - Also called Network Access Points (NAP)

- Regional networks:
  - smaller backbones that cover smaller geographical areas (e.g., cities or states)

- Point of presence (POP):
  - machine that is connected to the Internet

- Internet Service Providers (ISPs):
  - provide dial-up or direct access to POPs
Internet Connection Hierarchy

Private “peering” agreements between two backbone companies often bypass IXP

Colocation sites

ISP (for individuals)  Regional net  Big Business
  |  |  |
  POP POP POP
  |  |  |
  T1 T1 Cable modem

Small Business  Pgh employee  DC employee
  |  |  |
  POP POP POP
  |  |  |
  T1 POP DSL

Cable modem

T1
**IP Address Structure**

- **IP (V4) Address space divided into classes:**

  - **Class A**: 0 1 2 3 8 16 24 31
    - Net ID: 0
    - Host ID: 32
  - **Class B**: 1 0 8 16 24 31
    - Net ID: 10
    - Host ID: 24
  - **Class C**: 1 1 0 8 16 24 31
    - Net ID: 110
    - Host ID: 24
  - **Class D**: 1 1 1 0 8 16 24
    - Multicast address
  - **Class E**: 1 1 1 1 8 16 24
    - Reserved for experiments

- **Network ID Written in form w.x.y.z/n**
  - n = number of bits in host address
  - E.g., CMU written as 128.2.0.0/16
    - Class B address

- **Unrouted (private) IP addresses:**
  - 10.0.0.0/8 172.16.0.0/12 192.168.0.0/16
Evolution of Internet

■ Original Idea
  ▪ Every node on Internet would have unique IP address
    ▪ Everyone would be able to talk directly to everyone
  ▪ No secrecy or authentication
    ▪ Messages visible to routers and hosts on same LAN
    ▪ Possible to forge source field in packet header

■ Shortcomings
  ▪ There aren't enough IP addresses available
  ▪ Don't want everyone to have access or knowledge of all other hosts
  ▪ Security issues mandate secrecy & authentication
Evolution of Internet: Naming

- Dynamic address assignment
  - Most hosts don't need to have known address
    - Only those functioning as servers
  - DHCP (Dynamic Host Configuration Protocol)
    - Local ISP assigns address for temporary use

Example:
- Laptop at CMU (wired connection)
  - IP address 128.2.213.29 (bryant-tp4.cs.cmu.edu)
    - Assigned statically
- Laptop at home
  - IP address 192.168.1.5
  - Only valid within home network
Evolution of Internet: Firewalls

Firewalls
- Hides organizations nodes from rest of Internet
- Use local IP addresses within organization
- For external service, provides proxy service
  1. Client request: src=10.2.2.2, dest=216.99.99.99
  2. Firewall forwards: src=176.3.3.3, dest=216.99.99.99
  3. Server responds: src=216.99.99.99, dest=176.3.3.3
  4. Firewall forwards response: src=216.99.99.99, dest=10.2.2.2