TCP/IP and Socket Programming

CS3214 Instructors

Virginia Tech

November 21, 2024

VIRGINIA TECH

Outline

- Goal: obtain working knowledge of TCP/IP (+UDP), including IPv4/IPv6, to become productive with writing simple network applications
- Transport layer protocols: TCP and UDP
  - Use of ports
  - Demultiplexing in TCP/UDP
- IPv4 addressing & routing
  - including subnets & CIDR
- Protocol independence (IPv6)
- BSD *socket* API
  - including utility functions for DNS name resolution

VIRGINIA TECH

Transport and Network Layer

- Transport Layer Protocols: UDP and TCP
  - TCP: reliable data transmission
  - UDP: unreliable (best effort) data transmission
  - Port numbers are used to address applications
- Network Layer Protocols: IPv4 and IPv6
  - IP addresses are used to address hosts (*)
- Both protocols are designed to work with IP, hence the terms TCP/IP and UDP/IP

(*) technically, network interfaces - will explain difference shortly

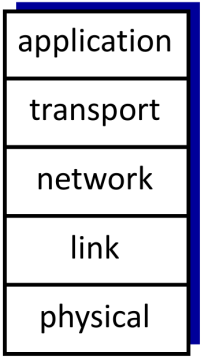


Figure 1: Internet Protocol Stack

User Datagram Protocol - UDP

- Specified in RFC 768 (1980)
- **simple**: specification is 2 pages
- **datagram oriented**: up to 64K messages
- **connectionless**: no connection setup required
- **unreliable**: best effort, makes no attempt to compensate for packet loss
- supports multicast

UDP datagram header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Length																Checksum															

Figure 2: Source: WikiPedia

VIRGINIA TECH

Transmission Control Protocol - TCP

Specification

RFCs: 793 (1981), 1122 (1989), and many subsequent ones, see 7414[1] for 2015 road map.

- **point-to-point**: one sender, one receiver
- **reliable, in-order byte stream**: no "message boundaries"
- **pipelined**: transmission proceeds even while partially unack'ed data
- **send & receive buffers**: to hold this data
- **full duplex data**: bi-directional data flow in same connection
- **connection-oriented**: handshaking (exchange of control msgs) before data exchange
- **flow controlled**: sender will not overwhelm receiver
- **congestion controlled**: protects the network

VIRGINIA TECH

TCP segment header

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved **0 0 0**				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																															
20 …	160 …	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) …																															

Figure 3: TCP Segment Header. Source: WikiPedia

VIRGINIA TECH

TCP/IP & UDP/IP Addressing/Demultiplexing

Question:

How does process A on host H_1 communicate with process B on host H_2?

- Each stream is characterized by a quadruple (A_s, P_s, A_d, P_d) where
  - A_s, A_d are source and destination addresses - either a 32-bit IPv4 address or a 128-bit IPv6 address, e.g. 172.217.9.196 or 2607:f8b0:4004:807::2004
  - P_s, P_d are 16-bit port numbers - there is one namespace per address + protocol combination, e.g. 80/tcp, 80/tcp6, 53/udp, 53/udp6. See `/etc/services` for commonly used port numbers
- Local vs remote/peer addresses are pairs (A_s, P_s) or (A_d, P_d) respectively, depending on perspective
- Demultiplexing (determining where to deliver incoming packets) requires full quadruple for TCP, but only (A_d, P_d) for UDP

VIRGINIA TECH

IP Addresses

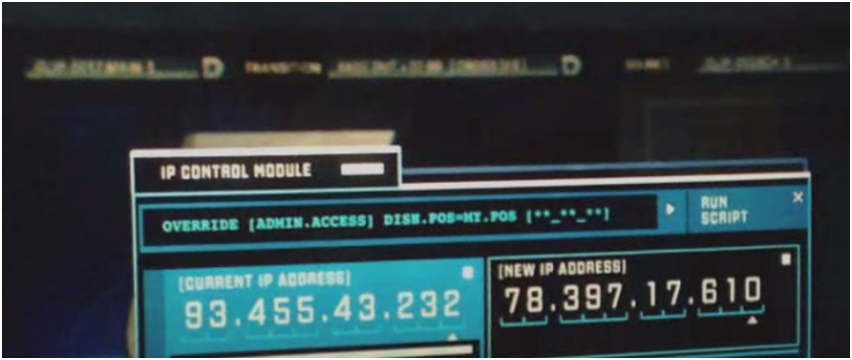


Figure 4: What's wrong with this picture? Source: http://i.imgur.com/zXR0qAN.png

VIRGINIA TECH

IPv4 Addressing

- IP addresses do not denote hosts, they denote interfaces (a host may have more than 1)
- Connected interfaces form a subnet whose addresses must share a common prefix
- Subnets are routing destinations
- No routing within subnet - can reach destination directly
- CIDR allows for up to 31 prefix bits:
  - 223.1.1.0/24 includes 223.1.1.0 – 223.1.1.255 (netmask 255.255.255.0)
  - 223.1.7.0/30 includes 223.1.7.0 – 223.1.7.3 (netmask 255.255.255.252)

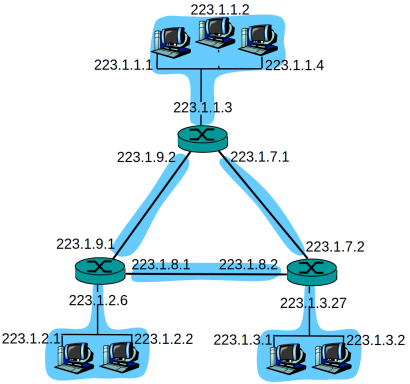


Figure 5: Subnetting in IPv4

VIRGINIA TECH

IPv4 Address Space Subdivision

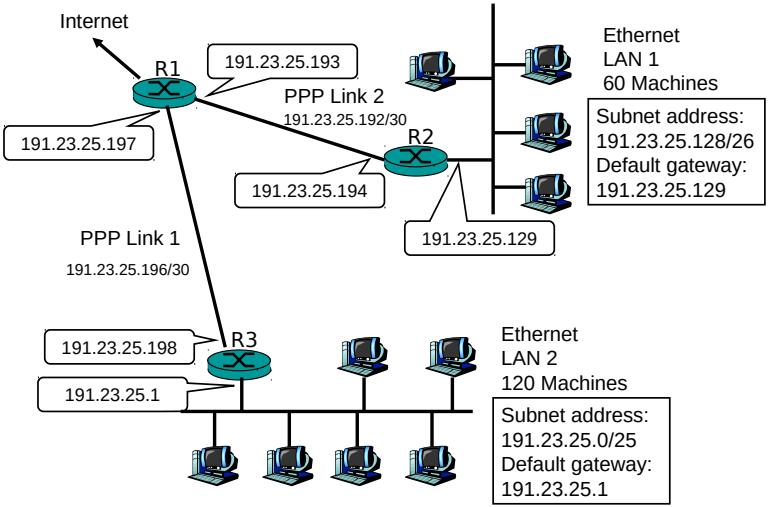
CS 5565 Exam Question

You are hired as a network administrator by a small company. You are given a small block of 256 addresses at 191.23.25.0/24.[a] You have to connect 2 LANs with 60/120 machines at 2 separate sites via PPP to an edge router at your ISP. Assign IP addresses to each subnet!

[a] Hypothetically. As of 2020, all available IPv4 address space is assigned, and this belongs to Telefônica Brasil

VIRGINIA TECH

IPv4 Address Space Subdivision: Solution



The Socket API

- first introduced in BSD 4.1 Unix (1981), now de facto standard on all platforms
- as a general interprocess communication (IPC) facility:
  - a host-local, application-created, OS-controlled interface (a "door") into which application process can both send and receive messages to/from another application process
- when used for network communication:
  - a door between application process and end-to-end transport protocol (UDP/TCP)
- in Unix, sockets are file descriptors, so `read(2)`, `write(2)`, `close(2)` and others work
- Bindings exist in many higher-level languages: e.g. `java.net.Socket`, Python `socket`

VIRGINIA TECH

UDP Socket API

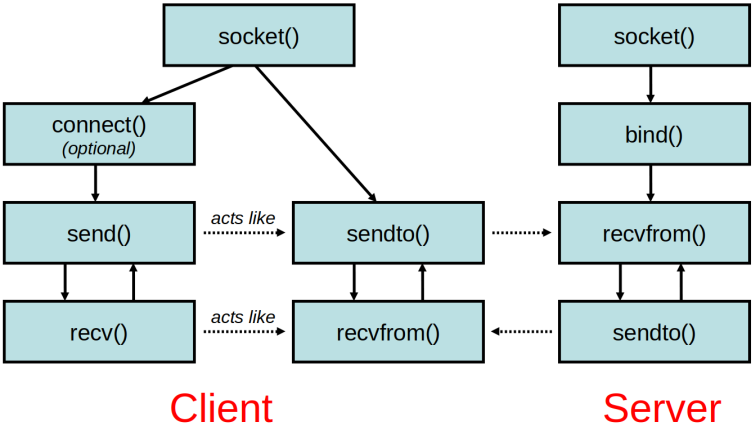


Figure 6: Socket API calls used in typical UDP communication scenario

VIRGINIA TECH

socket(2)

Usage:

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

- domain: `PF_INET`, `PF_UNIX`, `PF_INET6`, ...
- type: `SOCK_DGRAM` (for UDP), `SOCK_STREAM` (for TCP), ...
- protocol: 0 for Unspecified (or `IPPROTO_UDP` or `IPPROTO_TCP`)
- returns integer file descriptor
- entirely between process and OS – no network actions involved whatsoever
- man pages: ip(7), udp(7), tcp(7), socket(2), socket (7), unix(7) type "man 2 socket", "man 7 socket"

VIRGINIA TECH

bind(2)

Usage:

```
int bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen);
```

- sockfd: return by socket()
- my_addr: "socket address" - this is the local address (destination for receive, source for send)
- addrlen length of address (address is variable-sized data structure)
- "binds" (reserves, associates with) socket to (local) address specified in the protocol's namespace
- no information is transmitted over network
- one socket can be bound to one protocol/port, exceptions are
  1. multicast
  2. dual-bind same socket can bind to IPv4 and IPv6

VIRGINIA TECH

Address Family Polymorphism

```
struct sockaddr {     /* GENERIC TYPE, should be "abstract" */
    sa_family_t sa_family;  /* address family  */
    char sa_data[14];       /* address data  */
};

 /* This is the concrete "subtype" for IPv4 */
struct sockaddr_in {
  sa_family_t     sin_family; /* address family: AF_INET */
  u_int16_t       sin_port;   /* port in network byte order */
  struct in_addr  sin_addr;   /* internet address */
};

struct sockaddr_storage {   /* large enough to store addresses */
    sa_family_t sa_family;  /* address family  */
    char sa_data[?];        /* address data  */
};
```

VIRGINIA TECH

IPv4 vs IPv6 addresses

```c
/* Internet IPv4 address. */
struct in_addr {
    u_int32_t       s_addr;  /* address in network byte order */
};

/* IPv6 address */
struct in6_addr {
    union
      {
        uint8_t u6_addr8[16];
        uint16_t u6_addr16[8];
        uint32_t u6_addr32[4];
      } in6_u;
};
```

Good News

RFC 3493 functions for address manipulation mostly hide internal representations from the casual and professional socket programmer.

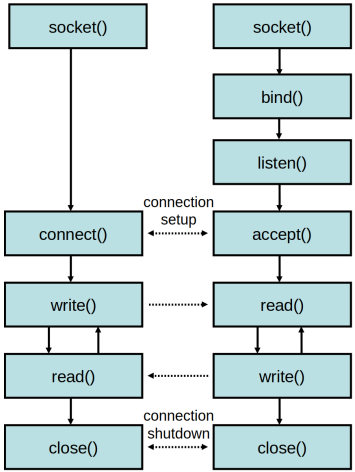
sendto(2), recvfrom(2), send(2), recv(2), connect(2)

```
ssize_t  sendto(int  s, const  void *buf, size_t len, int flags,
                const struct sockaddr *to, socklen_t tolen);
 ssize_t  recvfrom(int s, void *buf, size_t len, int flags,
                   struct sockaddr *from, socklen_t *fromlen);
```

- s, buf, len as in read/write
- flags: MSG_OOB, MSG_PEEK – mostly 0
- to/from are remote/peer addresses: where did the datagram come from, where should it be sent to
- NB: fromlen is value-result!
- Side note: can use connect(2) to set default address, then send(2)/recv(2).

VIRGINIA TECH

TCP Socket API Call Sequence



- Left: client ("connecting socket"), Right: server ("listening socket")
- Server may accept multiple clients via multiple calls to accept, either sequentially or concurrently
- Independent directions: `read(2)`/`write(2)` may be used in any order.
- `read(2)`/`write(2)` or `recv(2)`/`send(2)` may be used
- Not shown: `shutdown(2)` for shutting down one direction

VIRGINIA TECH

connect(2)

Usage:

```
int connect(int sockfd, const struct sockaddr *peeraddr, int addrlen);
```

- sockfd: returned by `socket()`
- peeraddr: peer address
- initiates handshake with server, sending SYN packet
- successful completion indicates successful handshake

VIRGINIA TECH

listen(2), accept(2)

Usage:

```
int listen(int s, int backlog);
int accept(int s, struct sockaddr *addr, int *addrlen);
```

- addr: accepted peer's (aka client) address
- `listen()` must precede call to `accept()`
  - No network action, but informs OS to start queuing connection requests
- `accept()` blocks until client is pending, then returns new socket representing connection to this client; the passed in socket is ready to accept more clients on subsequent calls

VIRGINIA TECH

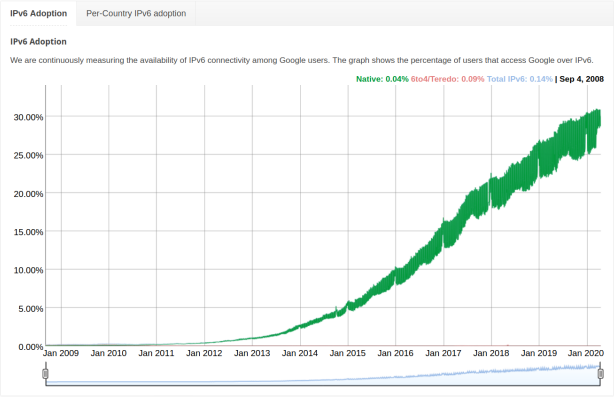
The IPv6 Challenge

- IPv4 provides only 4 billion addresses, leading to address space exhaustion
- IPv6 was designed as a successor in the 1990's
- ... but IPv6 is a separate network
  - A host may be connected via IPv4
  - ... or via IPv4 and IPv6
  - ... or only via IPv6
- Your network application must work in either case
  - Do not embed addresses or make assumptions about their size/format in your socket code
  - Let *system* tell you which address(es) you should use (as a client)/you should support (as a server)

VIRGINIA TECH

IPv6 Transition Plan

Servers provide both IPv4 and IPv6, clients prefer IPv6 to IPv4 when both are available, eventually IPv4 connections will die out ... will it happen?



IPv6 adoption among users accessing Google services, Feb 24 2020

VIRGINIA TECH

Protocol Independent Programming

```
int getaddrinfo(const char *node, const char *service,
                const struct addrinfo *hints, struct addrinfo **res);
```

- Use `getaddrinfo()` to obtain information about suitable address families and addresses
  - For servers to bind to (IPv4, or IPv6, or both): if `AI_PASSIVE` is set and `node == NULL`
  - For clients to connect to (based on DNS name or specified address notation); based on RFC 3484 (now RFC 6724) ordering
- Use `getnameinfo()` to transform addresses in printable form
- Mostly correct tutorial at http://www.akkadia.org/drepper/userapi-ipv6.html, except for pesky issue of how to support both families as a server
  - can use so-called dual-bind feature (with care, Linux-only)
  - portable solution is to use 2 separate sockets.

VIRGINIA TECH

References

[1] Martin Duke, Robert T. Braden, Wesley Eddy, Ethan Blanton, and Alexander Zimmermann.
A Roadmap for Transmission Control Protocol (TCP) Specification Documents.
RFC 7414, February 2015.

VIRGINIA TECH