High-Level UDP / TCP/IP

Network Programming

Overview

- UDP (User Datagram Protocol) is unreliable
  - Potentially delivered out of order (or not at all!)
  - Connectionless
- TCP (Transmission Control Protocol) provides reliability
  - In-order
  - Connection-oriented
- Both sit on top of another protocol

Internet Protocol (IP)

- IP is the network layer
- Essentially, responsible for host to host packet delivery (routing)
- Translation between multiple data link protocols such as ethernet, wifi, etc.

IP Datagrams

- IP provides connectionless, unreliable delivery of IP datagrams
- Connectionless: All datagrams are independent of each other
- Unreliable: no guarantee datagrams are even delivered, let alone ordered

IP Addresses

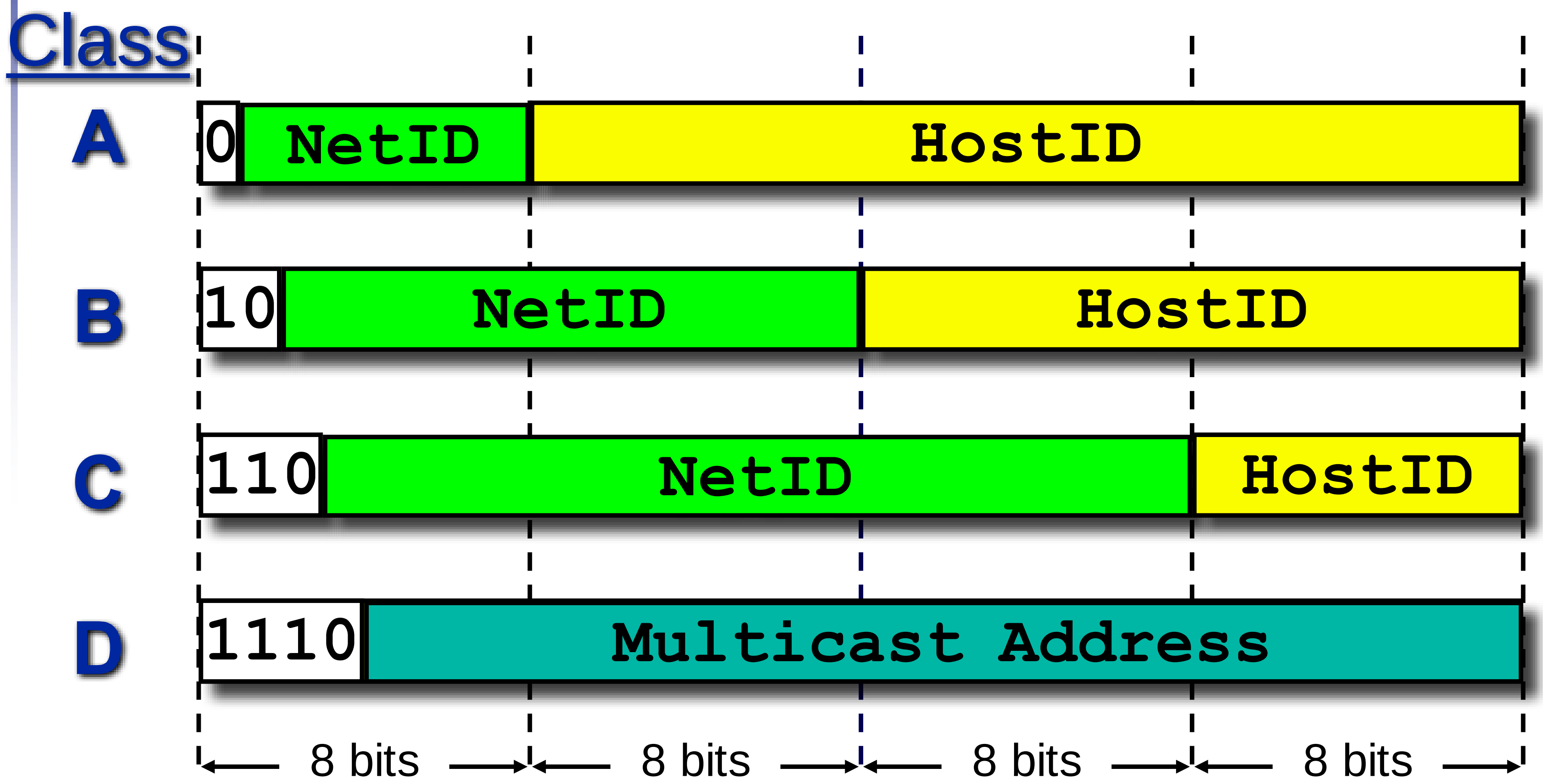
- IP at network layer but must be able to talk to other devices on different mediums! (ex: iPhone to wired server)
  - Also why MAC at different layer than IP

- Must provide some degree of *network* information
  - Allows for efficient routing of datagrams

IP Addresses

- IP addresses are logical addresses
- Four octets (32 bits) [IPv4]
- Includes a network portion and a host portion
- All IPs must be unique

IP Address Format



Ramifications

- Class A: 128 network IDs
  - 16M host IDs per network ID
- Class B: 16K network IDs
  - 64K host IDs per network ID
- Class C: 2M network IDs
  - 256 host IDs per network ID

Network / Host IDs

- Network IDs are assigned by a central authority
  - ICANN, IANA
- Host IDs are assigned locally by a systems administrator
- Network ID and host ID are used for routing purposes

IP Address Format

- IPs are often written in *dotted decimal* notation

- For example, 128.113.0.2 (www.rpi.edu)
  - 10000000.01110001.00000000.00000010

- RPI must have a class B address!
  - Leading digits are 10

Network / Host Addresses

- Hosts aren't assigned addresses, their network interface is
- Hosts may have multiple NICs
- If the network addresses are the same, they share the network

Broadcast / Network Addresses

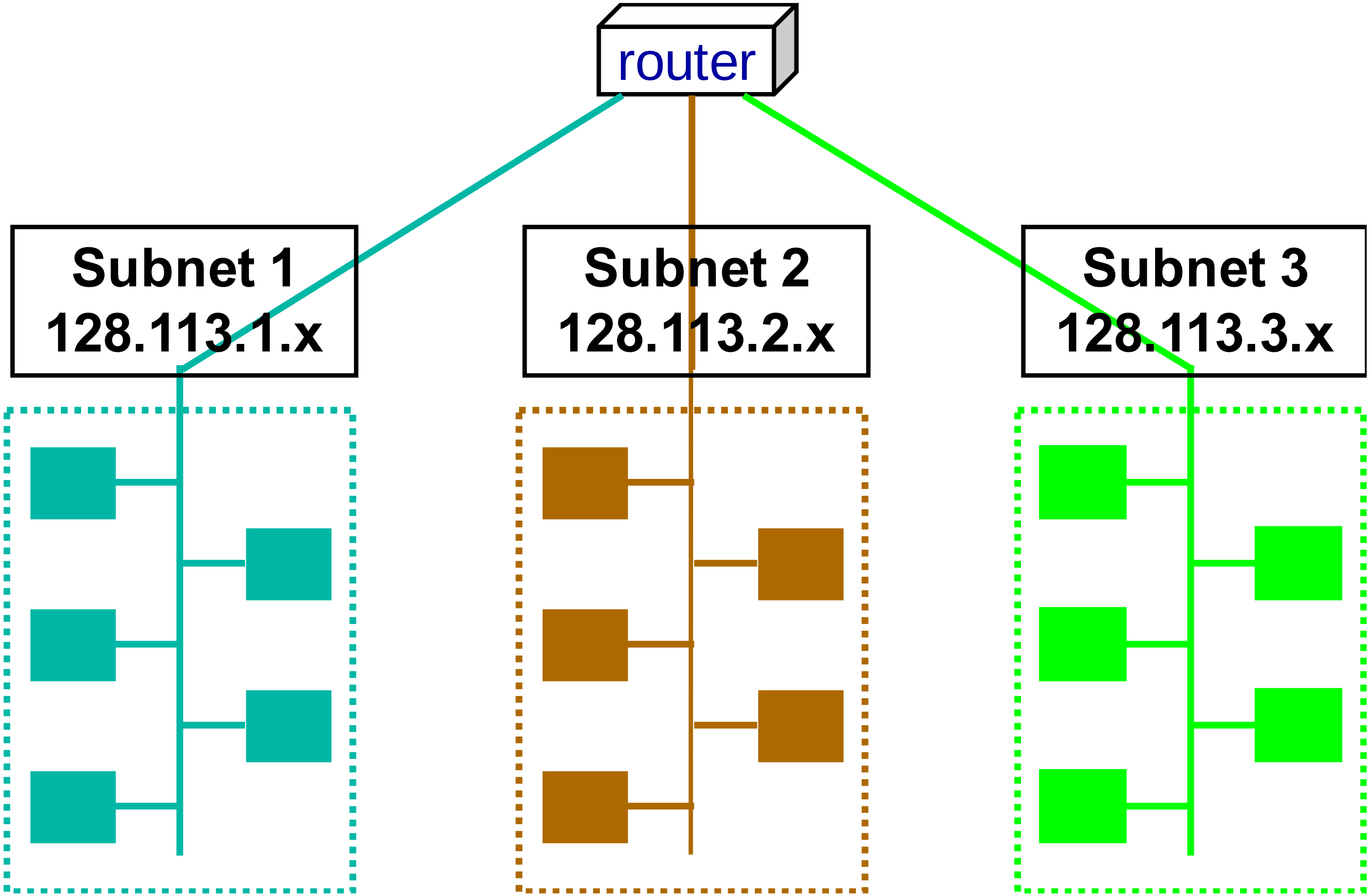
- Broadcast address: host ID all 1's
- Broadcasts may be implemented however the underlying layer sees fit
- Network address: host ID all 0's, refers to entire network

Subnet Addresses

- Organizations are able to further subdivide its available address space into “subnets”
- For example, clump nearby machines into their own subnet
  - Could also do this logically

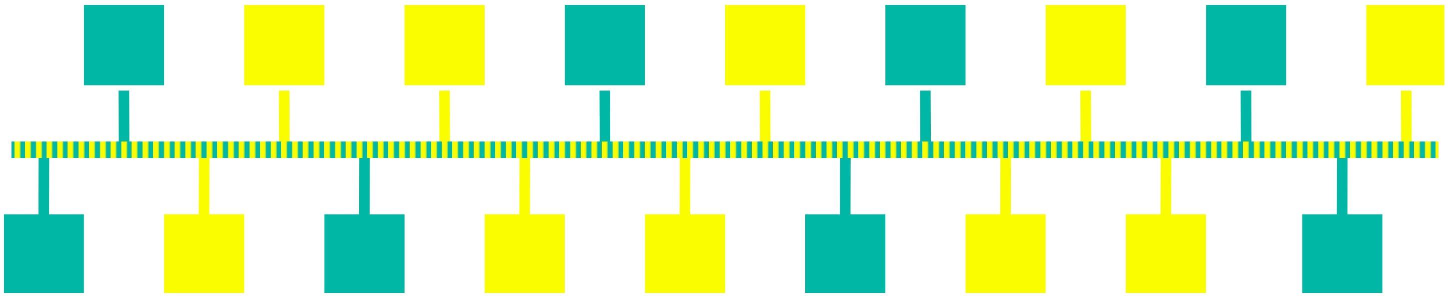
10	NetID	SubnetID	HostID

Subnetting


router
Subnet 1
128.113.1.x
Subnet 2
128.113.2.x
Subnet 3
128.113.3.x

Subnetting

- IP subnet broadcasts have host ID of all 1s
- Subnets can simplify routing
- Multiple subnets can share a single wire!



IP

- Connectionless delivery
  - Each datagram handled individually
- Unreliable
  - No guarantee
- Fragmentation + reassembly
  - Hardware MTU
- Routing
- Error detection

IP Datagram

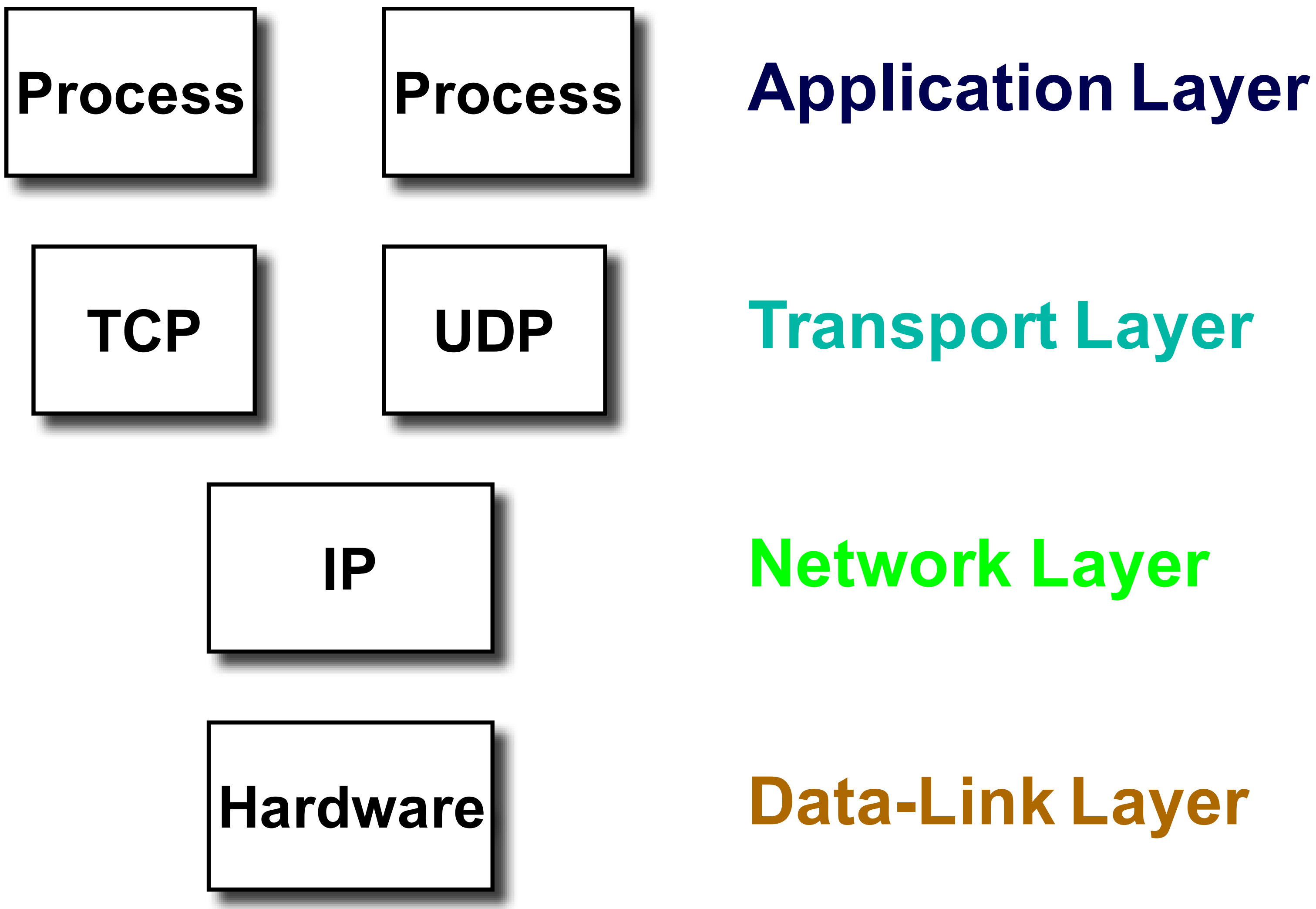
https://tools.ietf.org/html/rfc791#section-3.1

<table>
<tr><th colspan="2">1 byte</th><th colspan="2">1 byte</th><th colspan="2">1 byte</th><th colspan="2">1 byte</th></tr>
<tr><td>VERS</td><td>HL</td><td colspan="2">Service</td><td colspan="4">Fragment Length</td></tr>
<tr><td colspan="4">Datagram ID</td><td>FLAG</td><td colspan="3">Fragment Offset</td></tr>
<tr><td colspan="2">TTL</td><td colspan="2">Protocol</td><td colspan="4">Header Checksum</td></tr>
<tr><td colspan="8">Source Address</td></tr>
<tr><td colspan="8">Destination Address</td></tr>
<tr><td colspan="8">Options (if any)</td></tr>
<tr><td colspan="8">Data</td></tr>
</table>

Datagram Fragmentation

- Each fragment has same structure
- IP requires reassembly done at destination only, not at intermediate routers
- Any lost fragments require ICMP error message be sent and entire datagram discarded

TCP / UDP over IP



UDP

- UDP is a transport protocol
  - Communication between two processes

- UDP uses IP to deliver datagrams to the proper host

- Uses *ports* to provide additional specification

UDP Format

https://tools.ietf.org/html/rfc768

Source Port	Destination Port
Length	Checksum
Data	

TCP

- TCP is a transport protocol
- In addition to all things provided by UDP, TCP provides:
  - Reliability
  - Full-duplex
  - Connection-oriented
  - Byte Stream

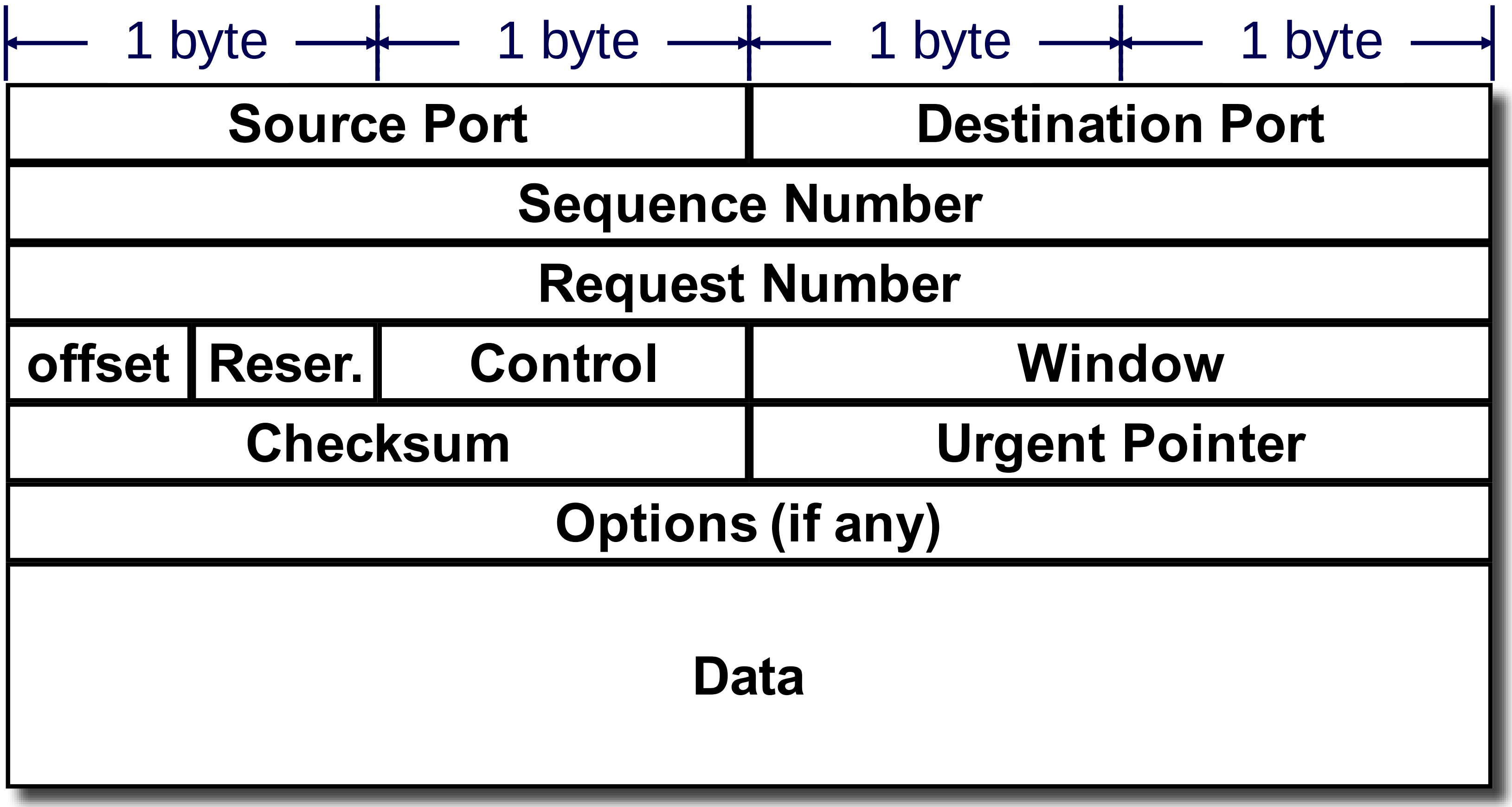
TCP Segments

- The chunk of data that TCP requests IP to transmit is called a *Segment*

- Each segment contains:
  - Data bytes from byte stream
  - Control information identifying data bytes

TCP Segment Format

https://tools.ietf.org/html/rfc793#section-3.1



UDP Sockets

Network Programming

Overview

- UDP (User Datagram Protocol) is unreliable
  - Potentially delivered out of order (or not at all!)
  - Connectionless
- TCP (Transmission Control Protocol) provides reliability
  - In-order
  - Connection-oriented

UDP

- UDP is a transport protocol
  - Communication between two processes
- UDP uses IP to deliver datagrams to the proper host
- Uses *ports* to provide additional specification

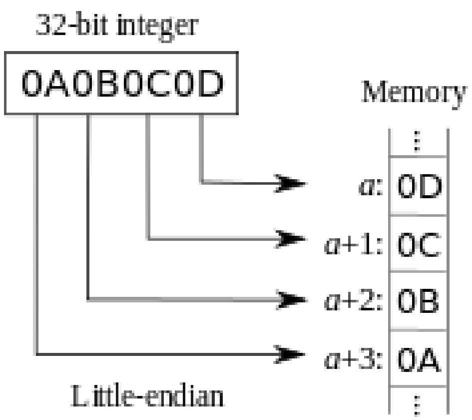
UDP Format

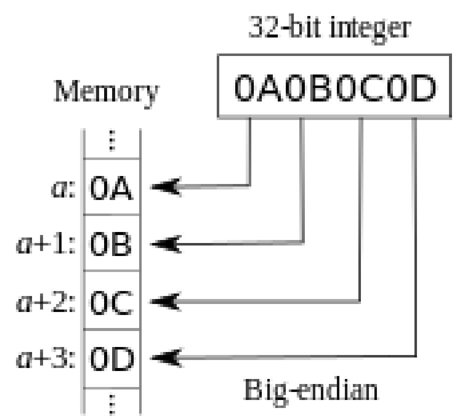
https://tools.ietf.org/html/rfc768

Source Port	Destination Port
Length	Checksum
Data	

But first… Endianness

- Some machines went big-endian (68K), others little-endian (x86)





- From Gulliver's Travels

Network Issues

- When hosts exchange single-byte data types, no problem
  - What about 32-bit word?
  - Are these the same value?
    - 0x00000001 and 0x01000000
  - Problem!

- Values sent from big-endian machine would be interpreted incorrectly on the little-endian machine!

Network Byte Order

- Network defines big-endian to be the byte order
  - May be different from *host byte order*
- Translation *always required*
  - Even on big-endian machines
  - How do you know what type of machine your code may be compiled on in the future?

Byte Order Functions

- `#include <netinet/in.h>`

- `uint16_t htons(uint16_t hs);`
- `uint32_t htonl(uint32_t hl);`

- `uint16_t ntohs(uint16_t ns);`
- `uint32_t ntohl(uint32_t nl);`

Sockets

- Berkeley sockets implementation, originally from 4.2BSD (1983!)
  - Effectively became POSIX sockets
- Building blocks for modern network-enabled programs
- Simple API

socket

- ```
  #include <sys/socket.h>
  ```
- ```
  int socket(int domain, int
     type, int protocol);
  ```

- Just creates an endpoint, nothing more!
- domain typically PF_INET / AF_INET
- type: SOCK_[STREAM,DGRAM,RAW]
- protocol: just use 0 for system default for given domain / type

bind

- `int bind(int fd, struct sockaddr *addr, socklen_t len);`

- fd: must be returned by socket()
- sa: sockaddr containing IP / port
- len: length of passed-in sockaddr

- Servers call bind upon startup

struct sockaddr_in

```
struct sockaddr_in {
    __uint8_t           sin_len;
    sa_family_t         sin_family;
    in_port_t           sin_port;
    struct  in_addr sin_addr;
    char                sin_zero[8];
};
```

```
struct in_addr {
    in_addr_t           s_addr;
};
```

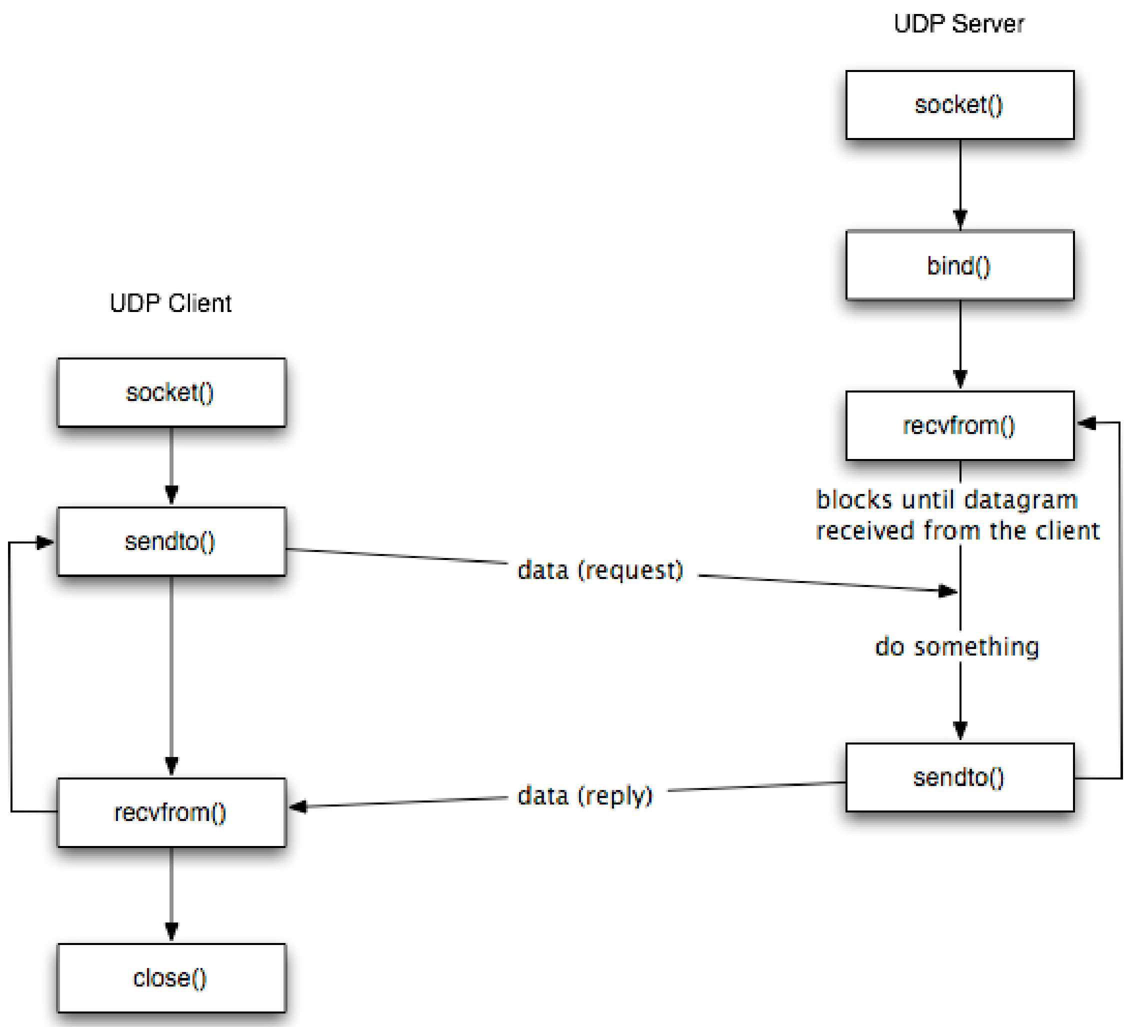
Typical usage

```
struct sockaddr_in saddr;

/* Zero out the memory */
bzero(&saddr, sizeof(saddr));

saddr.sin_family = PF_INET;
saddr.sin_port = htons(1234);
saddr.sin_addr.s_addr =
  htonl(INADDR_ANY);
```

Typical UDP client/server



recvfrom / sendto

```
#include <sys/socket.h>

ssize_t recvfrom(int fd, void
   *buf, size_t nbytes, int flags,
   struct sockaddr *from,
   socklen_t *len);

ssize_t sendto(int fd, void *buf,
   size_t nbytes, int flags,
   struct sockaddr *to, socklen_t len);
```

Oddities

- Sending 0 bytes is completely fine
  - 8 byte UDP header (no data)
- `recvfrom()` can return 0
  - Different from TCP where a 0 means peer has closed connection
- Both functions can also be used w/ TCP
  - But why???

udpserv01.c

```c
#include    "unp.h"

int
main(int argc, char **argv)
{
    int                 sockfd;
    struct sockaddr_in  servaddr, cliaddr;

    sockfd = Socket(AF_INET, SOCK_DGRAM, 0);

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family      = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port        = htons(SERV_PORT);

    Bind(sockfd, (SA *) &servaddr, sizeof(servaddr));

    dg_echo(sockfd, (SA *) &cliaddr, sizeof(cliaddr));
}
```

dg_echo.c

```
#include	"unp.h"

void
dg_echo(int sockfd, SA *pcliaddr, socklen_t clilen)
{
	int			n;
	socklen_t	len;
	char		mesg[MAXLINE];

	for ( ; ; ) {
		len = clilen;
		n = Recvfrom(sockfd, mesg, MAXLINE, 0, pcliaddr, &len);

		Sendto(sockfd, mesg, n, 0, pcliaddr, len);
	}
}
```

udpcli01.c

```c
#include    "unp.h"

int
main(int argc, char **argv)
{
    int                 sockfd;
    struct sockaddr_in  servaddr;

    if (argc != 2)
        err_quit("usage: udpcli <IPaddress>");

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(SERV_PORT);
    Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);

    sockfd = Socket(AF_INET, SOCK_DGRAM, 0);

    dg_cli(stdin, sockfd, (SA *) &servaddr, sizeof(servaddr));

    exit(0);
}
```

dg_cli.c

```c
#include     "unp.h"

void
dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
{
    int n;
    char    sendline[MAXLINE], recvline[MAXLINE + 1];

    while (Fgets(sendline, MAXLINE, fp) != NULL) {

        Sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);

        n = Recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);

        recvline[n] = 0;    /* null terminate */
        Fputs(recvline, stdout);
    }
}
```

Not reliable in any way!

- What if messages get lost?
  - Client datagram?
  - Server datagram?

- How do we add reliability?
  - Timeouts (and retransmissions)
  - Sequence numbers
  - We'll have an assignment about this later in the semester

UDP `connect()`

- We can call **`connect()`** on a UDP socket
  - No longer able to use **`sendto()`** but rather **`write()/send()`**
  - Similarly for **`recvfrom()`**; replace with **`recv()/recvmsg()`**

- May improve performance if communicating with same host repeatedly

UDP Clients: DNS

- Quicker, no three-way handshake required (we'll revisit this later)
- Connectionless means less burden on the nameservers (we may revisit DNS later)
- Loss isn't terrible, just send another query some time later

DNS Resource Records

- A: hostname -> IPv4 address
- AAAA: hostname -> IPv6 address
- PTR: IP address -> hostname
- MX: Mail Exchange
- CNAME: canonical name

Skipping: Zeroconf/Bonjour

- Builds on DNS (and other tech as well)
- SRV records added to DNS
  - https://tools.ietf.org/html/rfc2782
- Designed to simplify networking
- Skipping for now because configuring issues (avahi)