IP Addressing

Kevin Scrivnor

COMP 429

Spring 2023

Recall: traceroute from L04

- We were shown the path a packet takes from your machine to the destination
- Each IP address along the way was a router that forwards your packet
- The question for the next two weeks is how does the router know where to forward packets?
- This is the issue the network layer solves

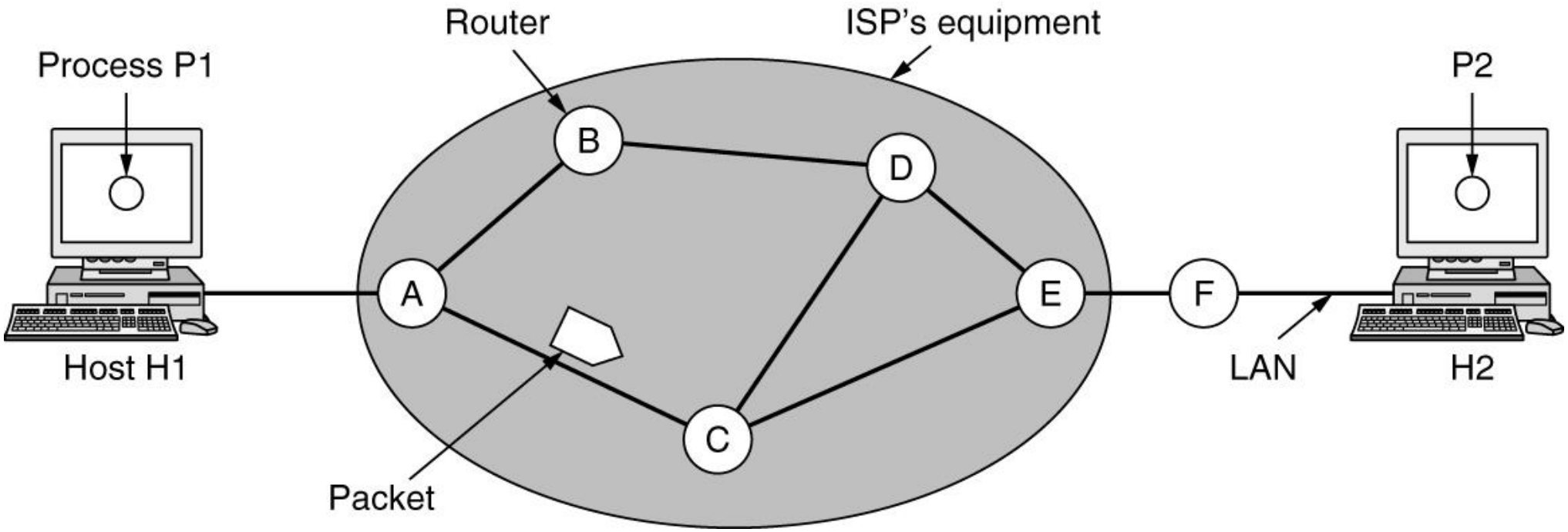
The Network Layer is concerned with getting all packets from the source to the destination

- This may require many “hops” as we saw with traceroute

- Contrast this to what the link layer solved
  - Modest goal of getting data across a wire (adjacent hosts)

- Network layer is the lowest layer in the stack that is end-to-end
  - Must be aware of all of the following
  - Topology of the network
  - Compute all the paths along the network
  - Choose the routes carefully
  - The source/destination are independently operated systems (ASes, Autonomous Systems)

Store and Forward Packet Switching



Services provided to the transport layer

- Goal
  - Independent of router technology
  - Transport layer should not know anything about the network (topology, number of routers, so on)
  - Network addresses should use a uniform numbering plan across all LANs and WANs

- Two Warring Factions
  - Connection-oriented vs. connection-less
  - Internet companies want connection-oriented because that's how the telephone system was designed

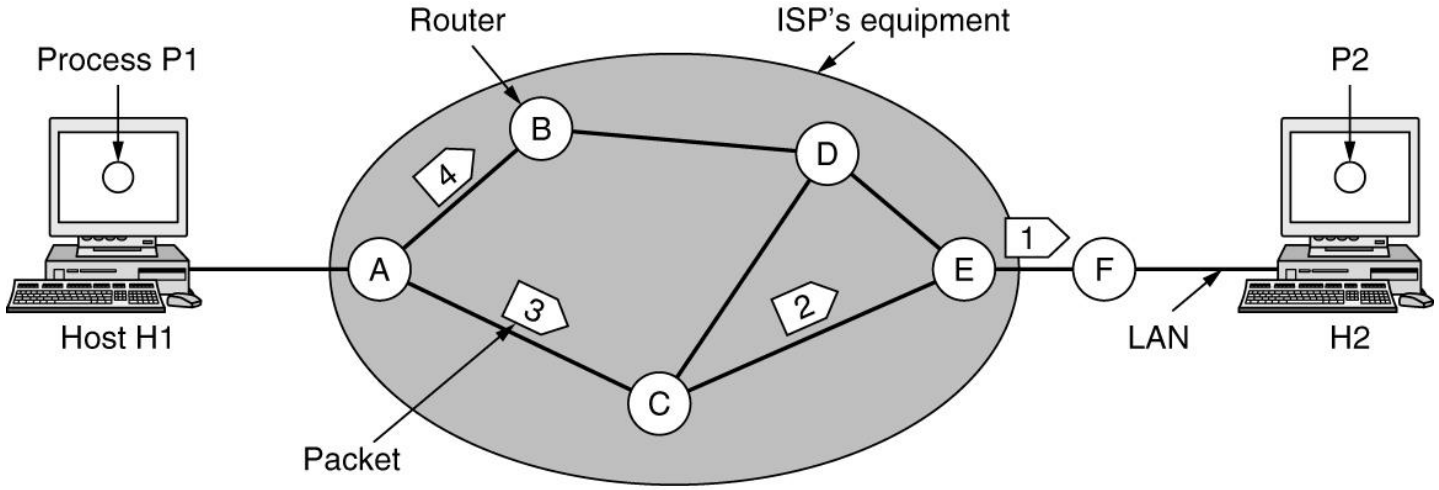
In the end, it's a little bit of both

- IP is the *Internet Protocol*
  - It is connectionless and it runs the Internet (winner)

- There still exists plenty of widely used connection-oriented tech
  - VLANs – Virtual LANs are widely used (on campus and in this lab)
  - Multiprotocol Label Switching is used
  - NAT is a weird kind of connection-oriented thing that we will see at the end of the lecture

Implementation of a connectionless service

- At the data link layer, we talked about *frames*
- At the network layer, we refer to *packets*
  - Sometimes you'll see the word *datagrams* (like a telegram)
  - Or a *Virtual Circuit (VC)* (like a telephone circuit)



A's table (initially)

Dest.	Line
A	–
B	B
C	C
D	B
E	C
F	C

A's table (later)

A	–
B	B
C	C
D	B
E	B
F	B

C's table

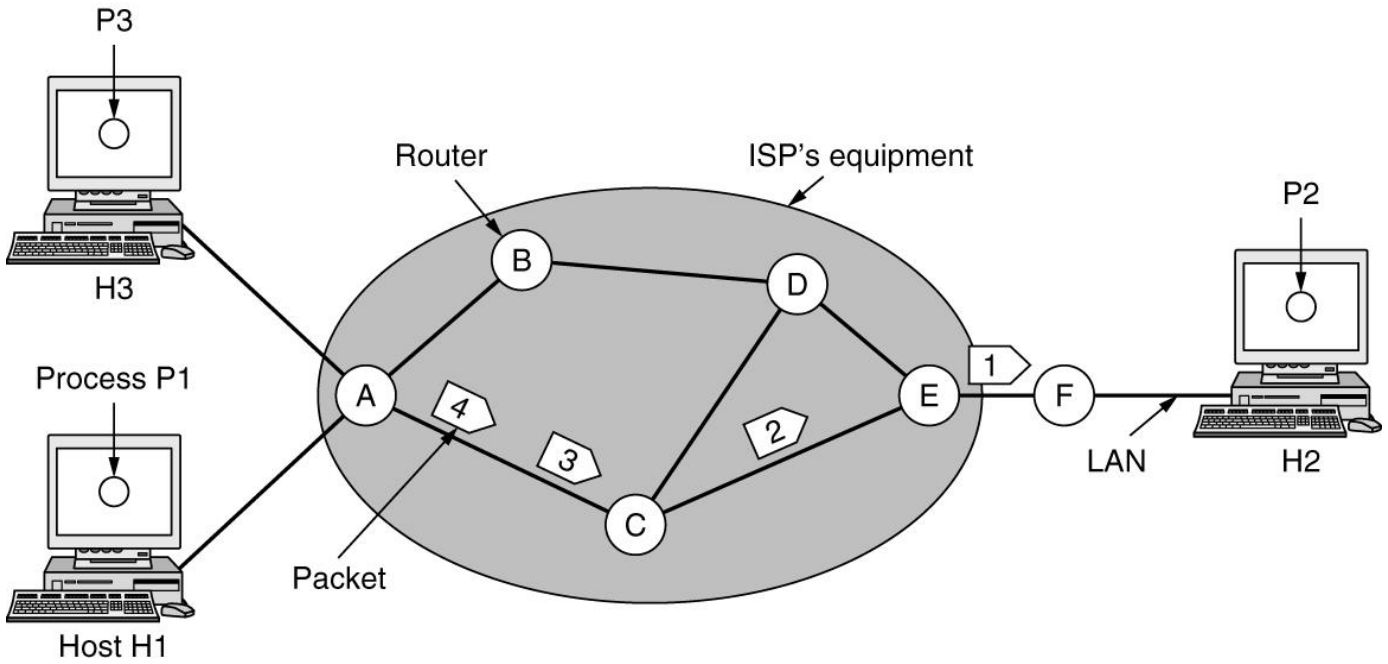
A	A
B	A
C	–
D	E
E	E
F	E

E's table

A	C
B	D
C	C
D	D
E	–
F	F

Implementation of a connection-oriented service

- Avoids having to choose a new route every time
- Each packet must contain some identifier
- Connections are established between the routers



Comparing virtual circuit & datagram networks*

Issue	Datagram network	Virtual-circuit network
Circuit setup	Not needed	Required
Addressing	Each packet contains the full source and destination address	Each packet contains a short VC number
State information	Routers do not hold state information about connections	Each VC requires router table space per connection
Routing	Each packet is routed independently	Route chosen when VC is set up; all packets follow it
Effect of router failures	None, except for packets lost during the crash	All VCs that passed through the failed router are terminated
Quality of service	Difficult	Easy if enough resources can be allocated in advance for each VC
Congestion control	Difficult	Easy if enough resources can be allocated in advance for each VC

The design of the past lead to the success the Internet is today

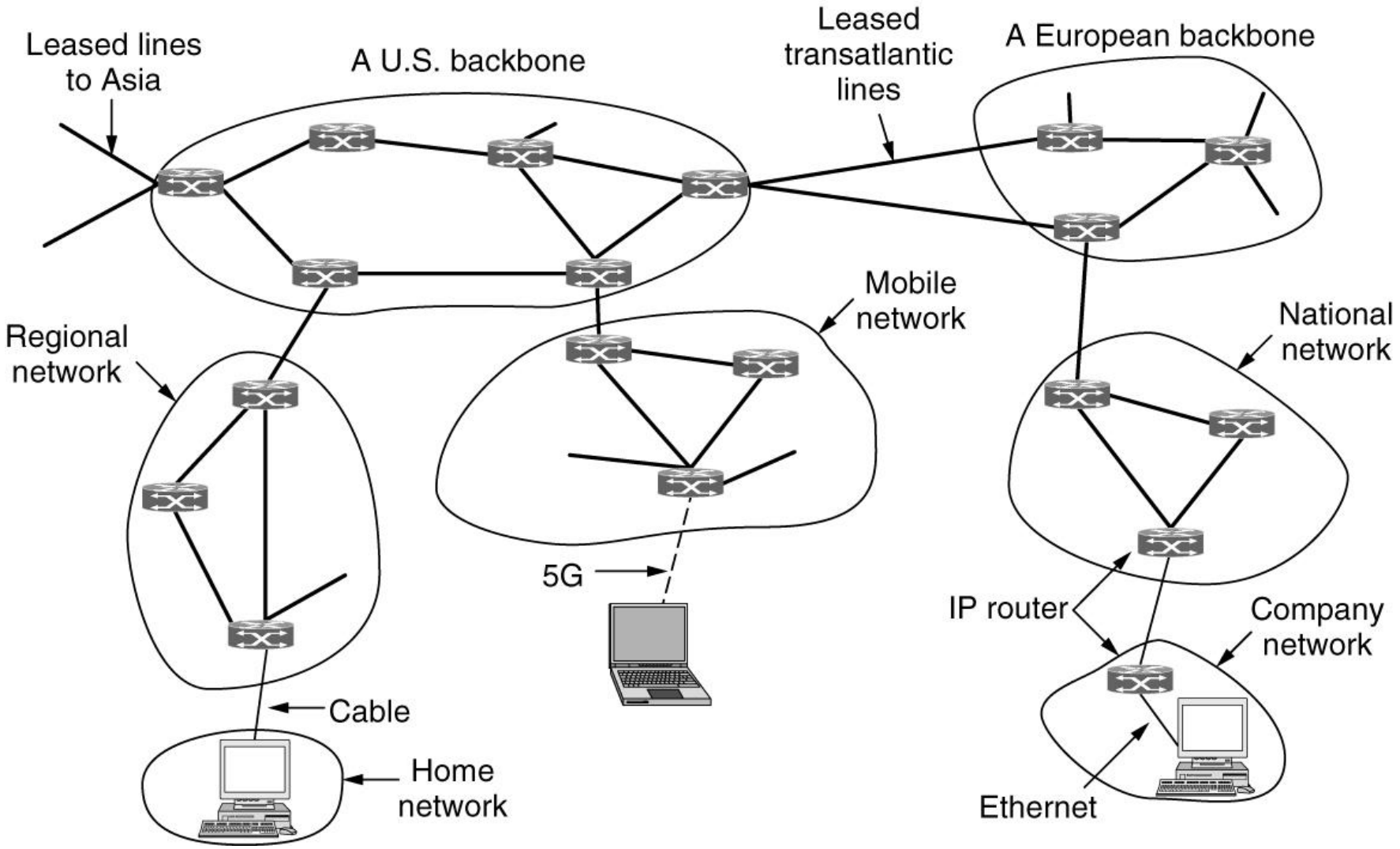
- Early work for the design of the network layer
  - (Clark, 1988) (Saltzer, 1984) and (RFC 1958)
- 10 principles of networking properly (most important first)

1. Make sure it works
   - Create multiple successful prototypes
   - Does it perform on the global Internet? Every region is different, hardware is different, software, etc.
2. Keep it simple
   - Leave out unnecessary features
3. Make choices clear
   - Leave no choices for implementation, make sure everything is defined

4. Exploit modularity
   - Use the protocol stacks
   - Independent layers, changes to one layer should not require change to another
5. Expect heterogeneity (je-knee-ity)
   - Simple, general, flexible
   - Many types of hardware, applications, software exist on a large network
6. Avoid static operators and parameters
   - Let the sender and receiver negotiate options
   - Vs. selecting/defining some fixed choice (you'll regret it later)
7. Look for a good design; it need not be perfect
   - People show up with weird requirements
   - Leave the good design alone, tell the people who want something strange to just figure it out themselves

8. Be strict when sending and tolerant when receiving
   - Send out perfectly crafted compliant packets
   - Expect to receive garbage on the other end
9. Think about scalability
   - Ideally load is spread evenly as possible
   - Millions and billions of hosts on the Internet
   - No centralized databases (always a bad idea)
10. Consider performance and cost
   - If it has bad performance, no one will use it
   - If it costs too much, no one will use it (no matter how great it is)

The Internet is a collection of networks or ASes

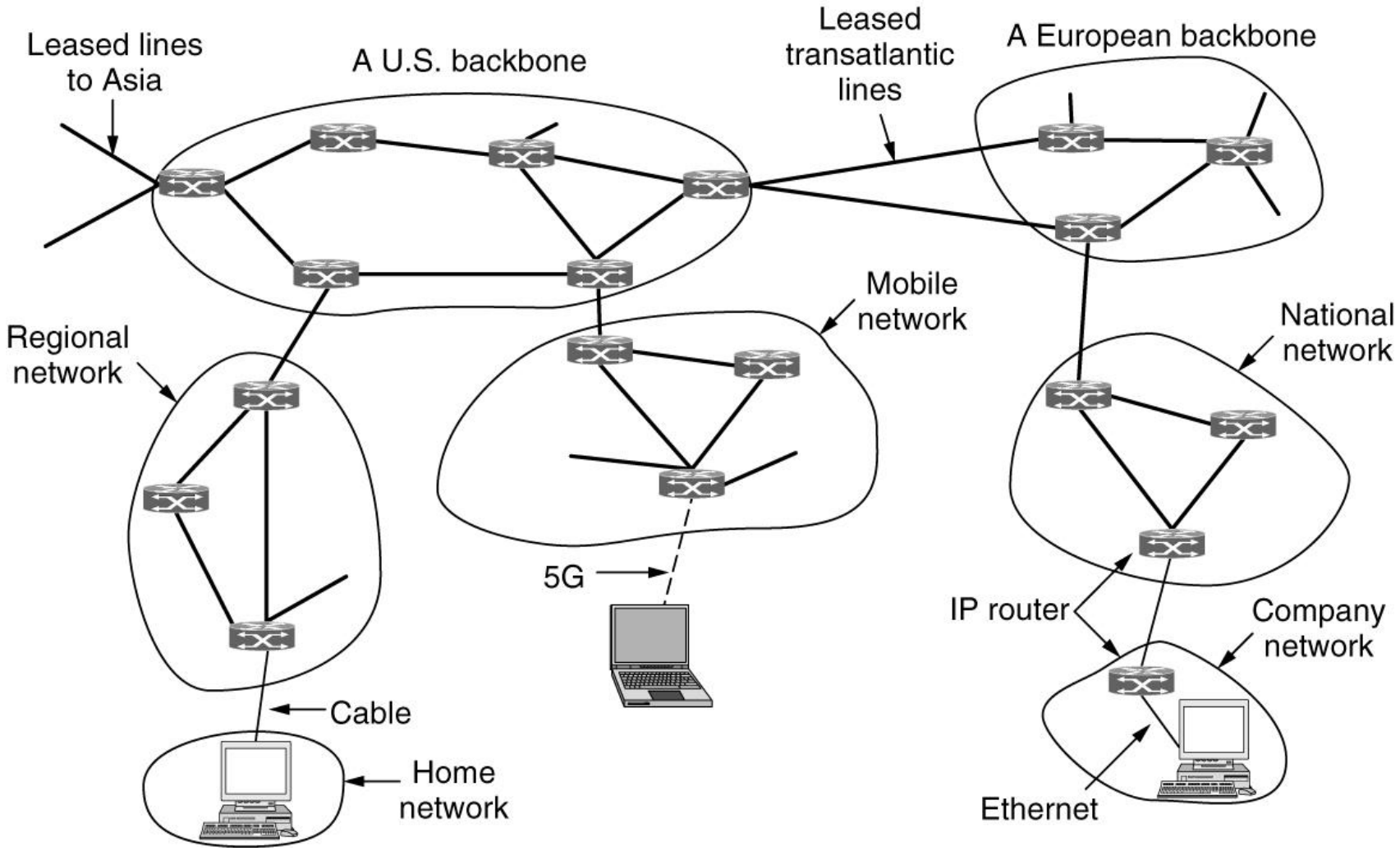


IP (Internet Protocol) is the glue that holds all this together

The Internet Protocol (IP)

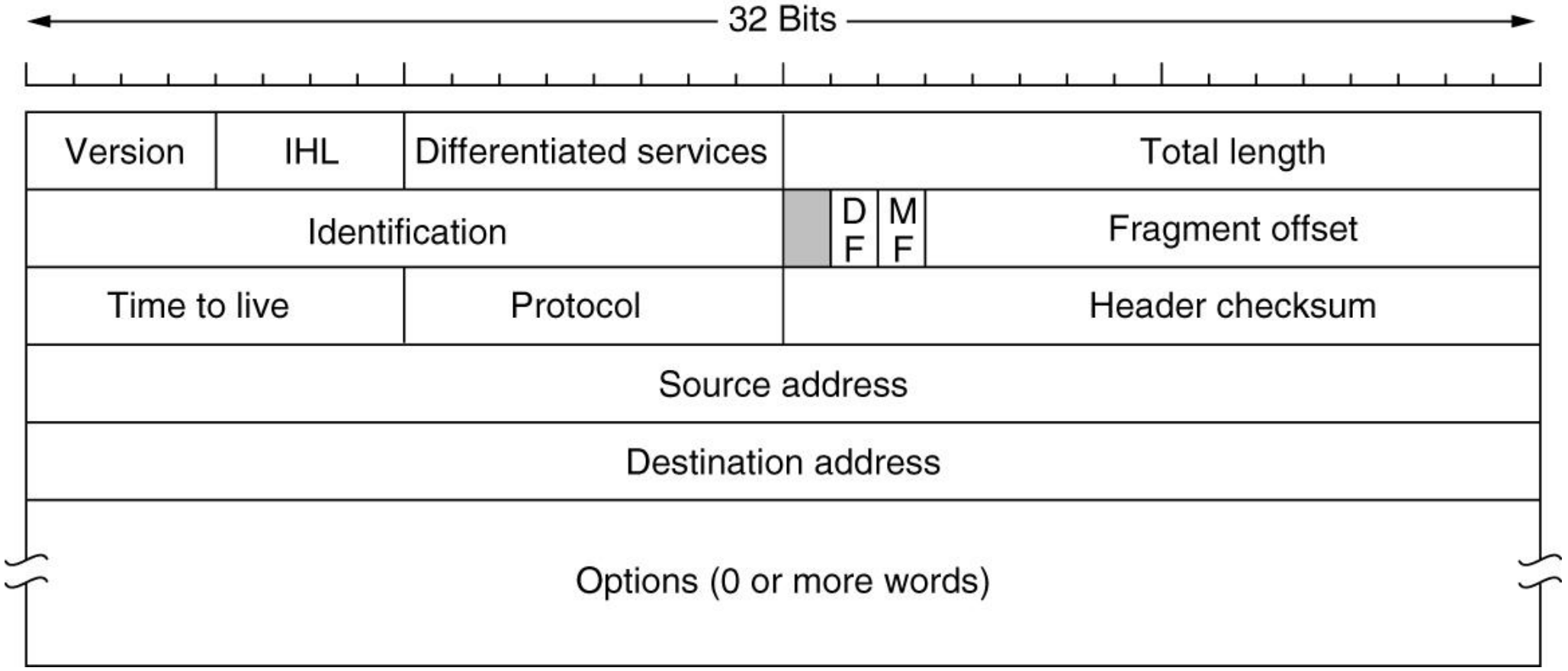
- A best effort protocol designed to deliver packets from source to destination without regard to what is in between
- Communication works as follows
  - Transport Layer breaks up data streams into packets
  - IP Packets
    - Theoretical maximum size of 64kb
    - In practice always 1500 bytes (size of 1 ethernet frame)
  - IP Routers
    - Forward each packet through the Internet path one router to the next
  - When the packets arrive, the network layer reassembles into the original datagram and hands it off to the transport layer

IP routes through many networks, with many redundant routes, running many different routing algorithms



IPv4 Header

- Version is given first, why?
- v5 was experimental, never widely used
- v6 created 10 years ago, eventually will be widely used



IP Addresses are 32 bit fields

- This gives us 2^32 unique IP addresses (only 4 billion)
- China and India each have > 2^31 people

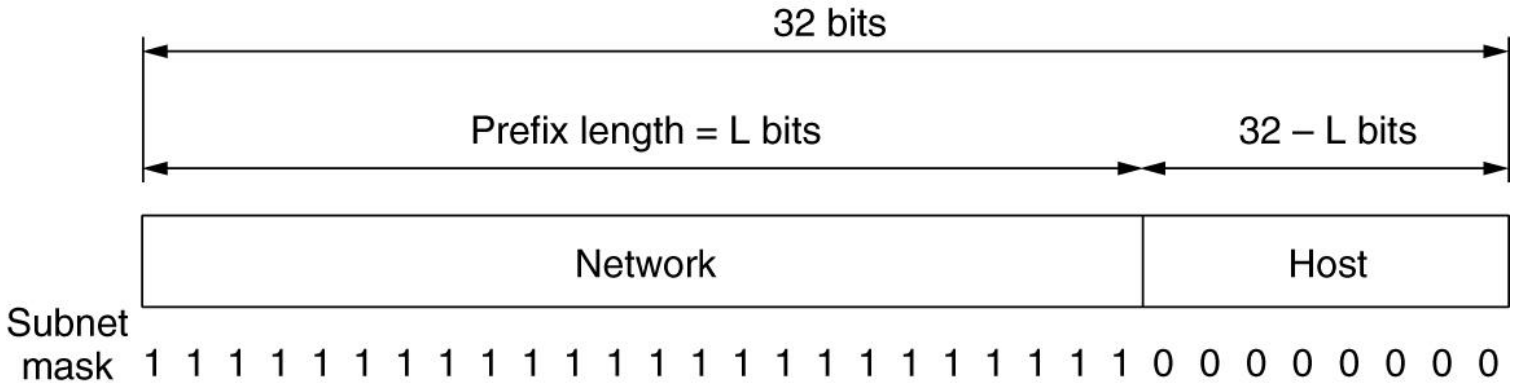
- Each network interface has its own IP address
  - (not each host, though generally each host does only have one IP)

- 0x80D00297 is an example of an IP address
  - 128.208.2.151 looks more familiar (dotted decimal format)

IP Addresses are hierarchical

- There is a variable length network portion (prefix) and then host portion

- 128.208.2.151/24
  - “slash 24” network, meaning the first 24 bits are reserved for the network, the last 8 are for the host
  - 128.208.2.0/24 is how we denote the subnet
  - Or with a netmask: 255.255.255.0 (first 24 bits are all 1’s)



Creating subnets makes routing much easier

- Routers forward packets based on the *network* portion of the IP address
- Relies on the fact that we don't need to necessarily care where an individual host is because knowing where the entire network is suffices
- Rather than a router having to have 4 billion entries for where every host is on the network, it just needs to know generally where subnets are
- Brings routing table entries down to about 200,000-300,000

There are some disadvantages of course…

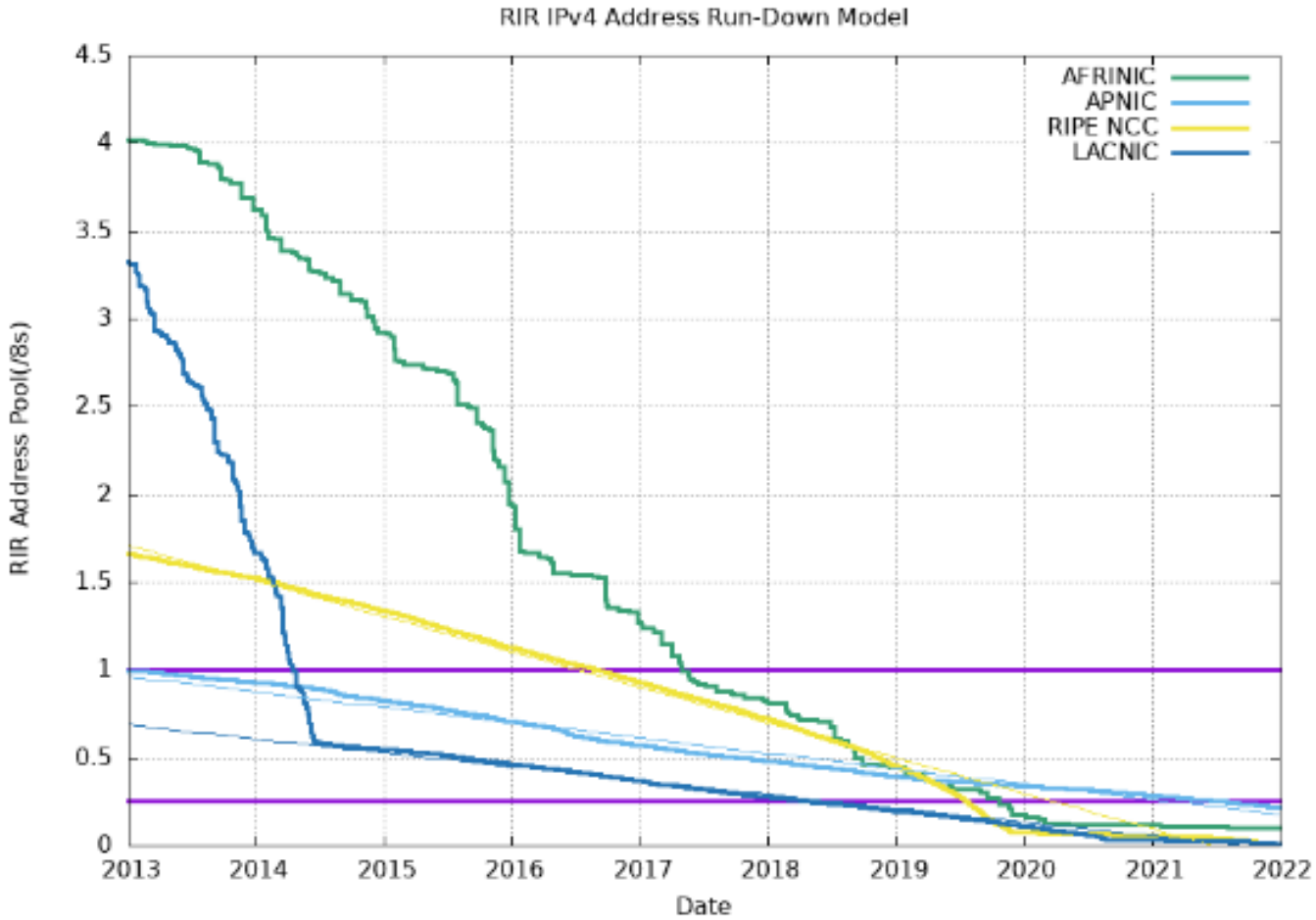
- IP becomes dependent upon geographical location
  - This makes mobile IP much harder
  - Phones moving between towers
  - Laptop moving between classrooms

- Hierarchy can be wasteful if mismanaged
  - Great pressure to carefully assign blocks of IP addresses
  - Managed globally by ICANN (in California)
  - Internet Corporation for Assigned Names and Numbers
  - Delegates addresses to regional authorities
    - NA, Asia, Africa, South America, so on

We are out of IP addresses to give away

- And have been for a while

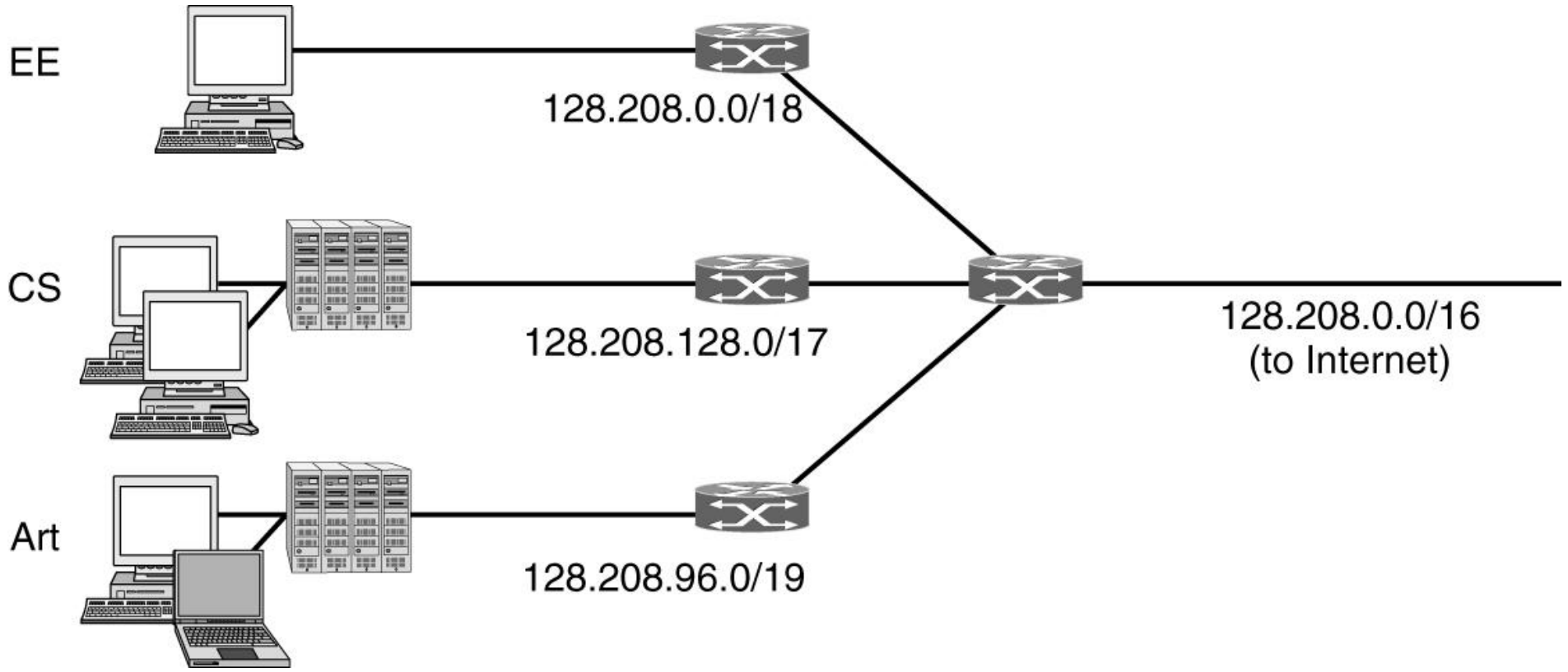
RIR	Projected Exhaustion Date	Remaining Addresses in RIR Pool (/8s)
APNIC:	**19-Apr-2011** (actual)	0.1577
RIPE NCC:	**14-Sep-2012** (actual)	0.0004
LACNIC:	**10-Jun-2014** (actual)	
ARIN:	**24 Sep-2015** (actual)	0.0000
AFRINIC:	**31-Dec--1**	0.0873



Projection of consumption of Remaining RIR Address Pools

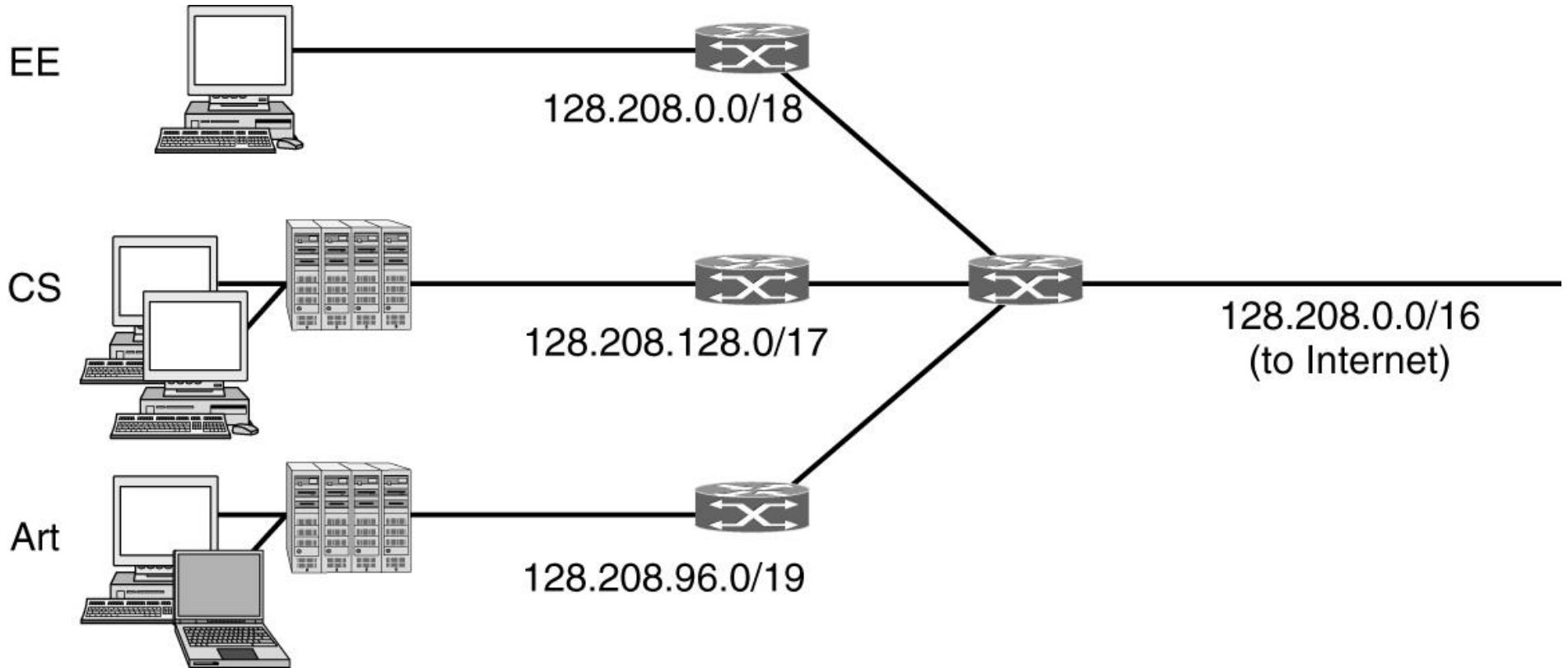
Subnetting breaks up larger networks into more manageable ones

- Question: how does router know how to forward packets?v



AND with subnet mask, look for matching prefix

- 128.208.2.151
- AND with /17, no match
- AND with /18, match! Forward that way



Classless Interdomain Routing (CIDR)

- Pronounced “cider”
- Without the hierarchical nature of subnetting, routing tables would need 4 billion entries

- Though, it’s harder to see why from your own computer

```
$ ip route
default via 10.0.2.2 dev enp0s3 src 10.0.2.15
10.0.2.0/24 dev enp0s3 src 10.0.2.15
192.168.56.0/24 dev enp0s8 src 192.168.56.100
```

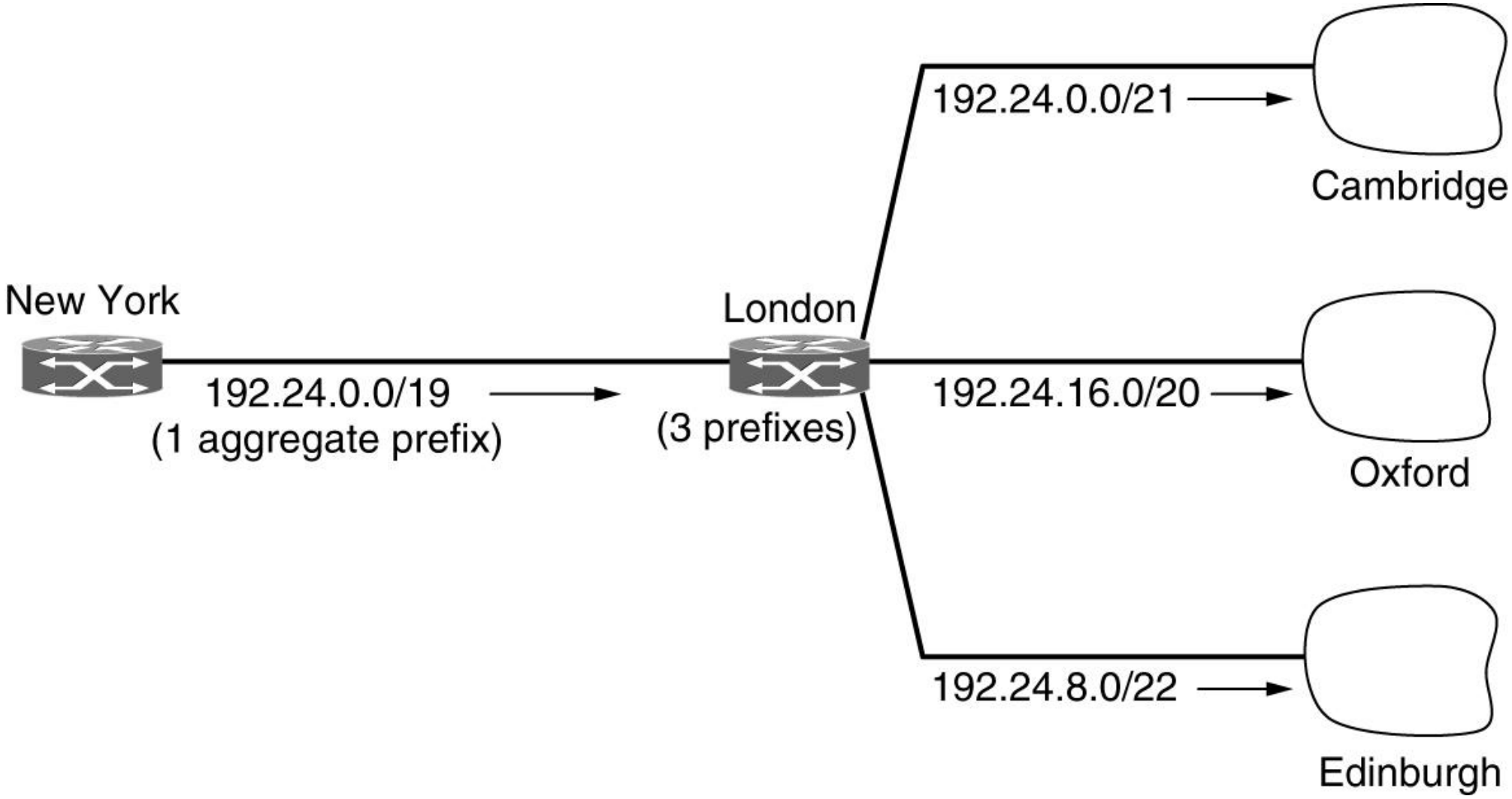
Your computer just needs to get packets to your router

- Then your router only needs to get packets to the ISP router
- But the ISP router needs to get packets to EVERYWHERE on the Internet…

- This is called the default-free-zone
  - There is no longer a “default” route
  - These routers have extremely specialized hardware
  - Forwards up to 1 million packets per second

- Solution is similar to subnetting
  - Supernetting, combining multiple prefixes into larger ones

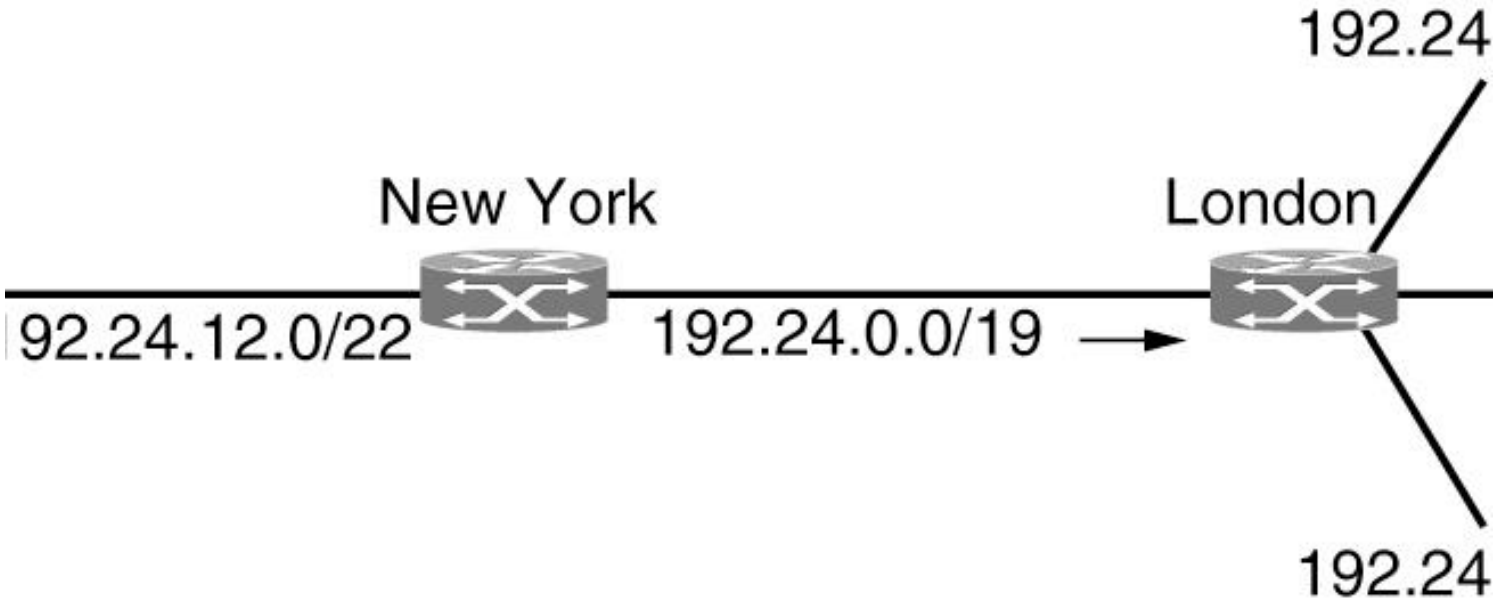
Supernetting



Conflicts

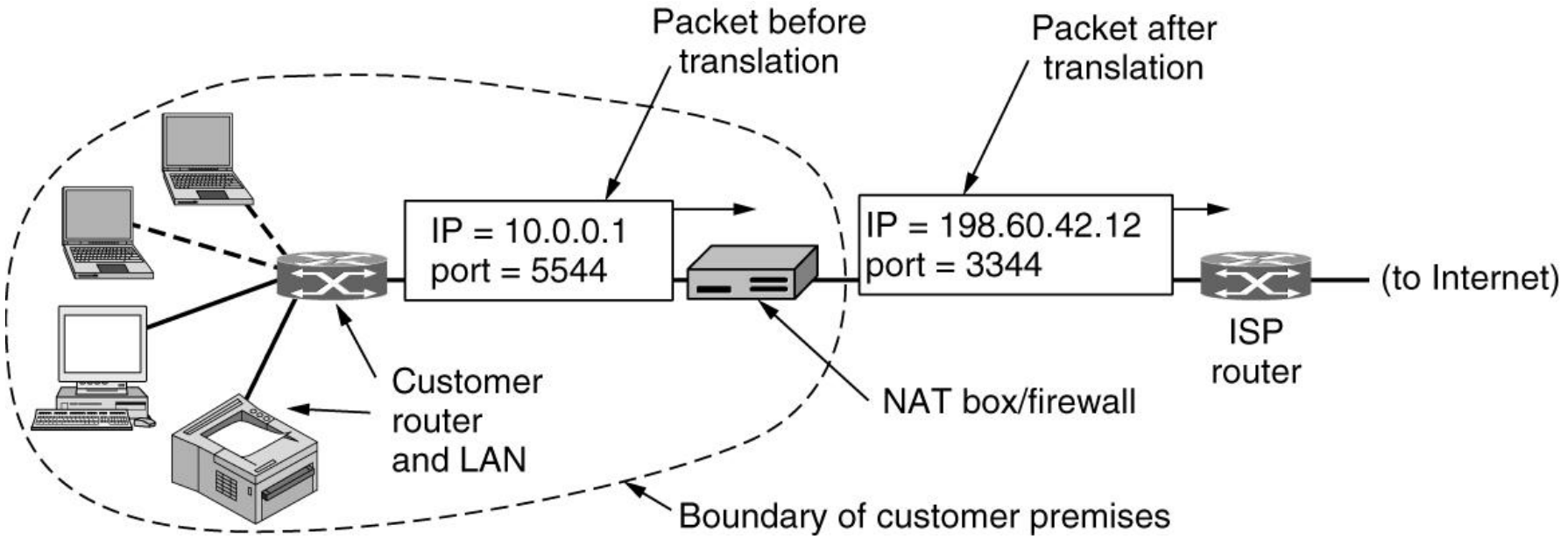
- Say somewhere in LA has IP in the subnet 192.24.12.0/22
  - Technically, both /19 and /22 match here, which route to take?
  - Router always takes the **longest matching prefix**

  - Very complex algorithms do this quickly
  - VSLI chips have the routing algorithms encoded into the hardware itself



The theory of running out of IP addresses is happening now

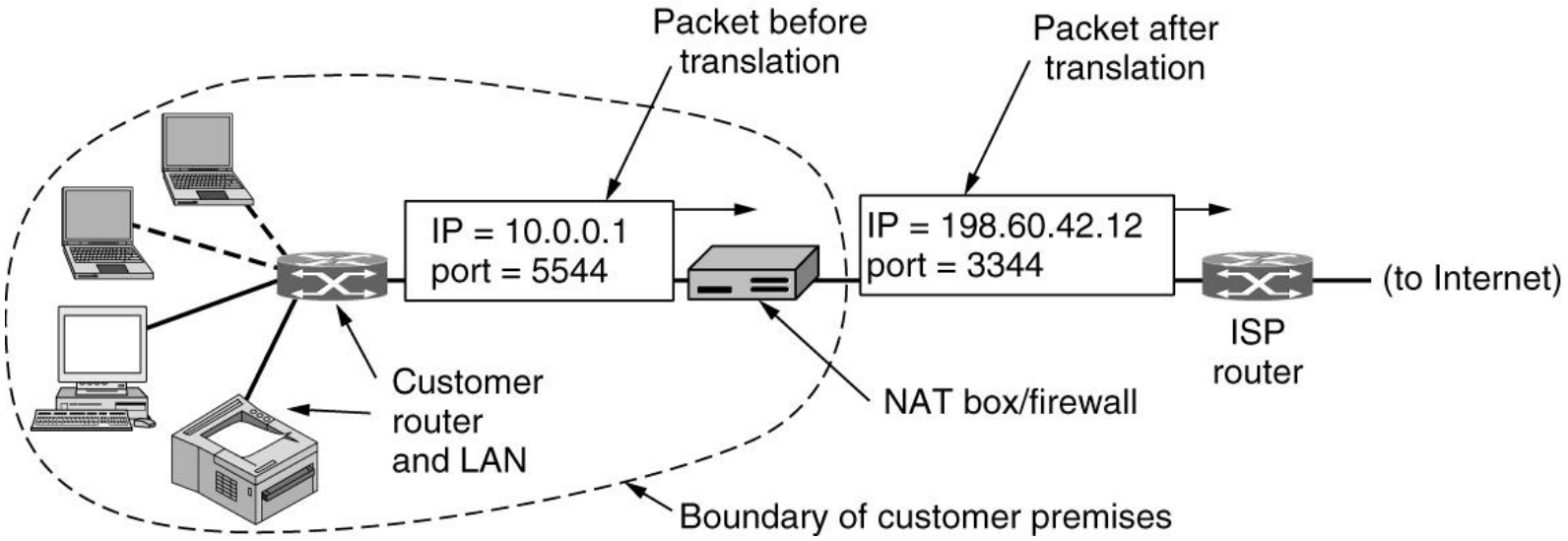
- If every computer needed a globally unique IP address, there would not be enough
- Since IPv6 is still “rolling out”
- We get by using NAT (Network Address Translation)



NAT

- IP does not have any extra fields
- However, most traffic is TCP or UDP
  - Which provides NAT a 16-bit field (port) to mess with
- Solves part of the problem, mostly works, adds some security in a way

- But violates the core beliefs from the beginning of the lecture



NAT Violations

- Violates the architectural model of IP
  - Every machine should have a unique IP
  - Thousands (if not millions) of machines have 192.168.1.1
- Breaks end-to-end connectivity model of the Internet
  - TCP has to connect to the outside of the network in order to establish the connection
  - With NAT, there is no way into a network behind a NAT'd gateway
- NAT makes the Internet a weird connection-oriented network
  - Every NAT router must maintain a state mapping each connection
  - (Port number and IP address) for the return data
- Violates the fundamental layering principle
  - Networking layer problem using transport layer solution (port #)

There’s more

- Only TCP and UDP will work with NAT
  - QUIC had to be built on top of UDP for this reason
- Applications sometimes use multiple connections or specific UDP ports
  - NAT breaks VoIP and makes FTP harder
- Uses the 16 bit port mapping
  - Limiting the ability to only serve up to 61,440 machines on a network
  - (65,536 without the 4096 reserved ports)

Socket Programming: UDP

Command: ip

```
kscrivnor@uvm:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:e4:a8:f8 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
       valid_lft 86141sec preferred_lft 86141sec
    inet6 fe80::387c:2b1a:74d1:d9a0/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:d6:05:e3:ee brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
```

- lo: loopback interface, where local network traffic resides (localhost)
- enp0s3: ethernet, the NAT'd network that the VM gets internet from
- docker0: local docker network for container traffic

Command: ss

- Socket statistics, the new version of netstat
- Every connection is associated with a socket 5-tuple
  - (protocol, source IP, source port, destination IP, destination port)
  - (TCP, 10.0.2.15, 55620, 138.197.192.78, 80)

```
kscrivnor@uvm:~$ ss -t4
State      Recv-Q      Send-Q            Local Address:Port              Peer Address:Port      Process
ESTAB      0           0                     10.0.2.15:55620           138.197.192.78:http
kscrivnor@uvm:~$ ss -t4n
State      Recv-Q      Send-Q            Local Address:Port              Peer Address:Port      Process
ESTAB      0           0                     10.0.2.15:55620           138.197.192.78:80
```

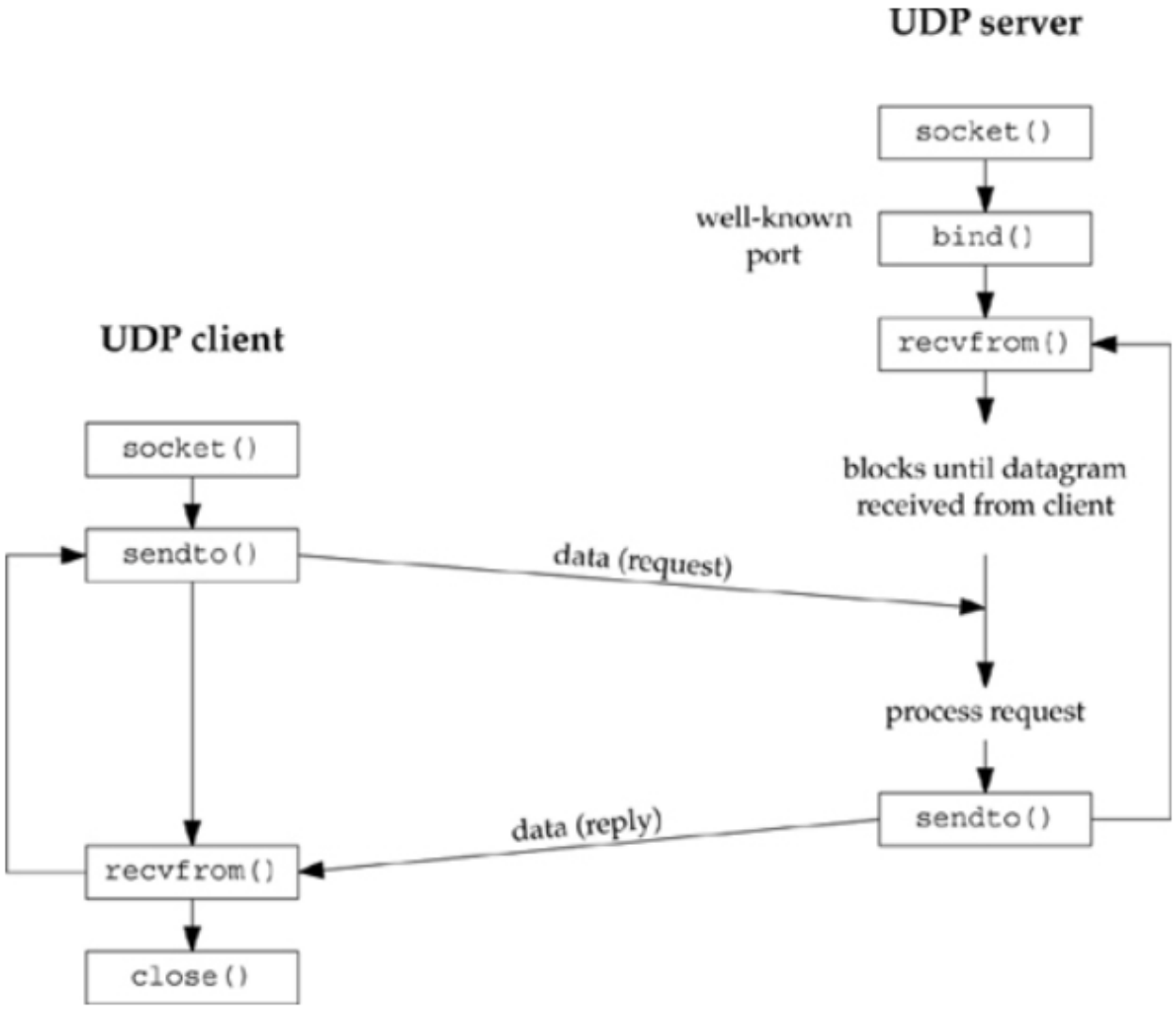
- Here I am connected to Scrivnor.cikeys.com (138.197.192.78) on port 80
- -t TCP sockets only, as opposed to –u (UDP sockets only)
- -4 Show IPv4 sockets
- -n show port numbers, rather than protocol (http vs 80)

Socket Example, but with NAT

- Demo + drawing

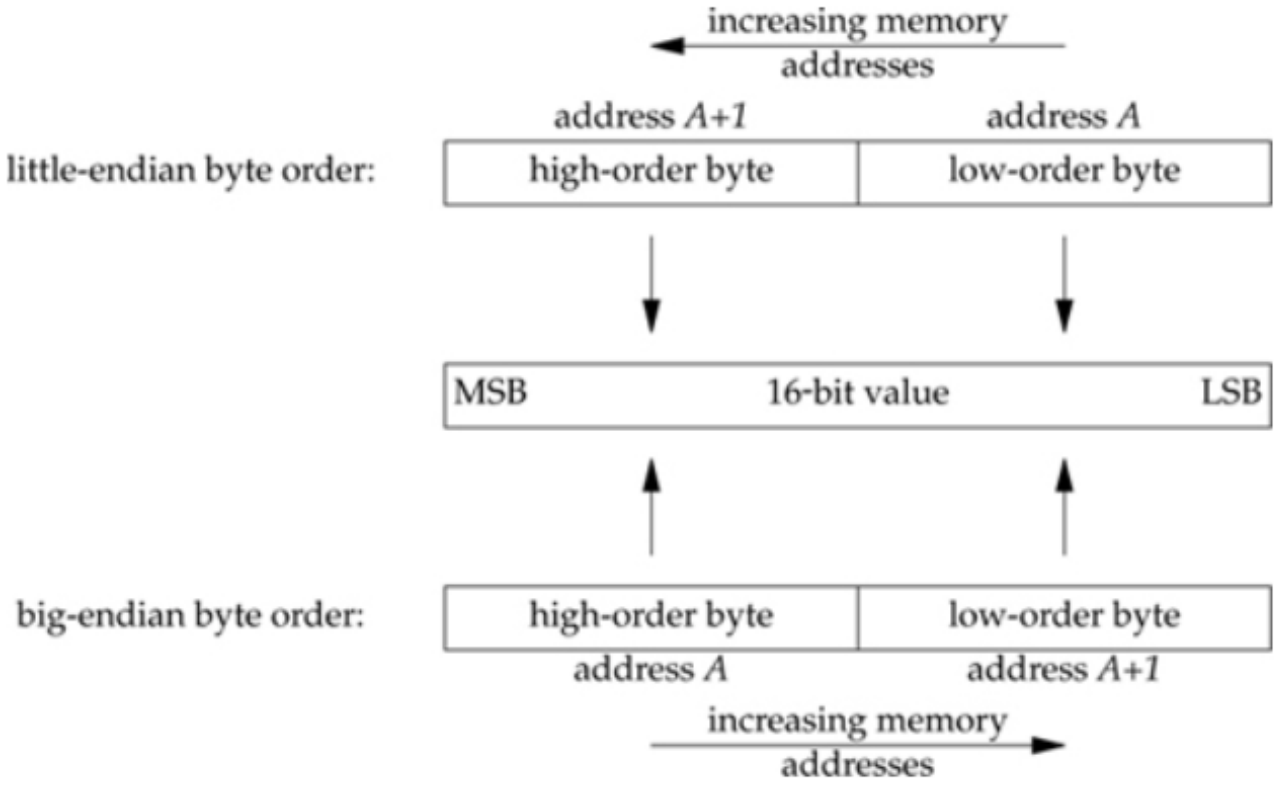
Flow of a UDP applications

- socket()
  - Creates a new file descriptor for communication, like open()
- bind()
  - Attach the socket to an IP address and port number to listen on
- sendto(), recvfrom()
  - Read and write to the socket
  - Reading can be a blocking call
- close()
  - Close the socket descriptor



Byte ordering: endianess

- Consider the two bytes: 0x1234
- Little-endian: 0x3412
- Big-endian: 0x1234
- Remember the difference
  - Little numbers first
  - Big numbers first
    - 0x1200 is bigger than 0x0034
- Internet protocols use big endian
- Intel and Apple Silicon both use little endian



Dealing with the endianness differences

- Host to Network byte ordering
  - htons() - read “host to network short” (16 bits)
  - htonl() - read “host to network long” (32 bits)

- Network to Host byte ordering
  - ntohs()
  - ntohl()

- Aside: the word byte not existing in older networking documentation
  - Octet is used in place, meaning 8 bits
  - A byte back then, wasn’t necessarily 8 bits on all machines like it is now

UDP Echo server: socket

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

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

```
// create socket
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
if( sockfd < 0 ) {
    perror("SOCKET CREATE ERROR");
    exit(EXIT_FAILURE);
}
```

- AF_INET, meaning, we will be using IPv4
- SOCK_DGRAM, datagram socket
- Protocol is 0 because UDP is implied by the socket type

UDP Echo server: bind

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

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

```
struct sockaddr_in {
    sa_family_t    sin_family; /* address family: AF_INET */
    in_port_t      sin_port;   /* port in network byte order */
    struct in_addr sin_addr;   /* internet address */
};

/* Internet address */
struct in_addr {
    uint32_t       s_addr;     /* address in network byte order */
};
```

```
// bind to address
struct sockaddr_in srvaddr = {};
srvaddr.sin_family = AF_INET;
srvaddr.sin_addr.s_addr = htonl(INADDR_ANY);
srvaddr.sin_port = htons(1337);
if( bind(sockfd, (struct sockaddr*) &srvaddr, sizeof(srvaddr)) < 0 ) {
    perror("SERVER BIND ERROR");
    exit(EXIT_FAILURE);
}
```

- AF_INET, again is IPv4
- INADDR_ANY means bind to basically 0.0.0.0
- 1337 is our port number we will listen on

UDP echo server: comm.

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

ssize_t recvfrom(int socket, void *restrict buffer, size_t length,
    int flags, struct sockaddr *restrict address,
    socklen_t *restrict address_len);
```

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

ssize_t sendto(int socket, const void *message, size_t length,
    int flags, const struct sockaddr *dest_addr,
    socklen_t dest_len);
```

```
// echo forever
char buf[MAX_BUF];
int n;
struct sockaddr_in cliaddr = {};
socklen_t clilen = sizeof(cliaddr);
for(;;) {
    n = recvfrom(sockfd, buf, MAX_BUF, 0, (struct sockaddr*) &cliaddr, &clilen);
    sendto(sockfd, buf, n, 0, (struct sockaddr*) &cliaddr, clilen);
}
```