LINUX

CNO

Programming

ACTP

ADVANCED CYBER TRAINING PROGRAM

NETWORKS

CNO CORE MODULE

ManTech Advanced Systems International, Inc.

Unauthorized Disclosure Prohibited by Trade Secrets Act: 18 U.S.C. § 1905

ManTech

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

Series Overview

ManTech

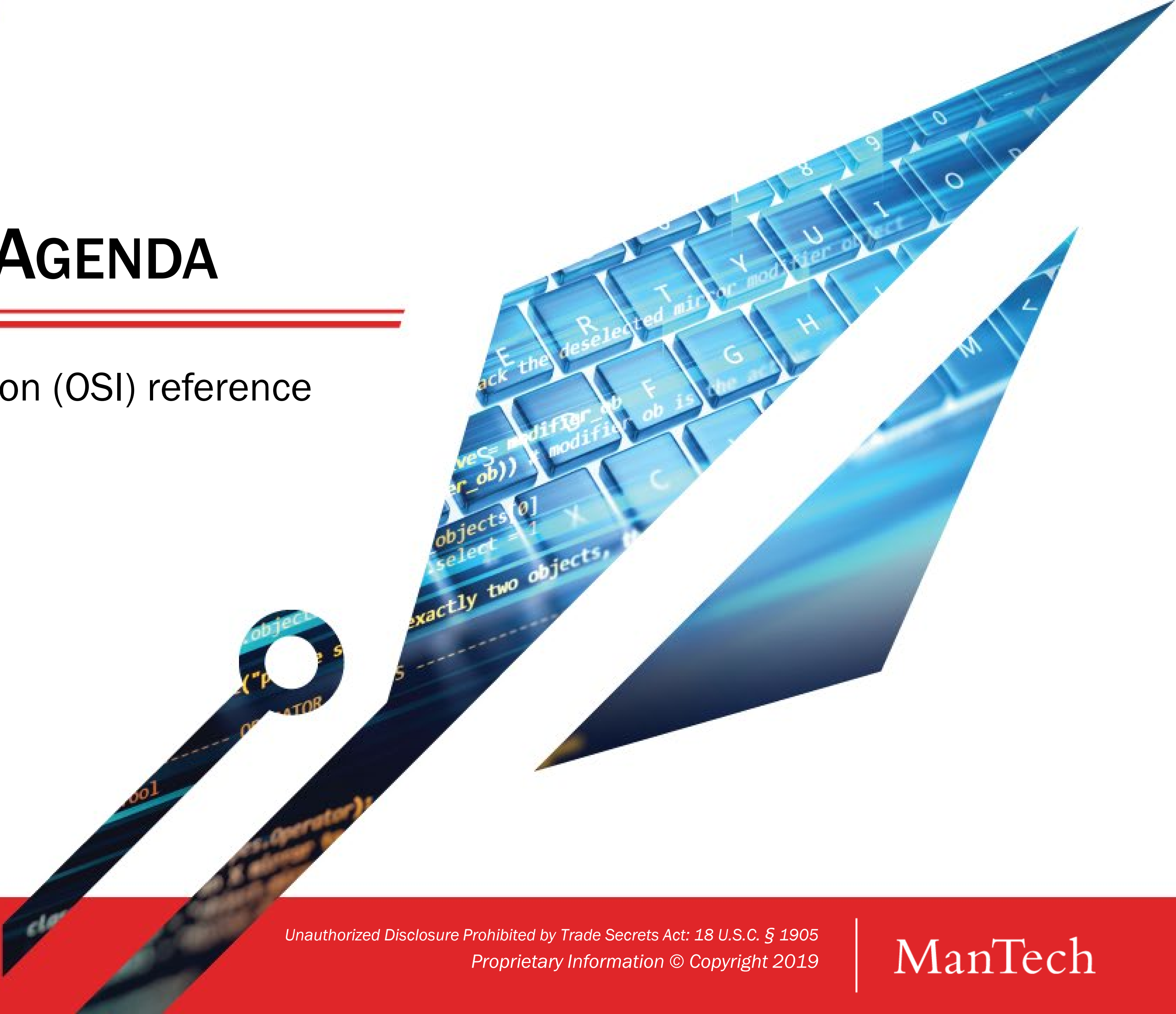
NETWORK
SERIES

- Series Outcome
  - After the Networks Series the student will have a conceptual foundation of the **network protocol stack**, significant **protocols** and how they **function in relation** to each other
  - The student will be able to develop basic network related tools
  - The student will also have a better understanding of IPv6
- Assessments
  - Final Assessment comprised of multiple choice questions:
    - Both knowledge-based and questions contingent upon the successful completion of integrated lab(s)
    - Minimum score of 80%

ACTP
ADVANCED CYBER TRAINING PROGRAM

NETWORK SERIES AGENDA

- Open Systems Interconnection (OSI) reference model
- Reference Documents
- Data Link Layer
- Network Layer
- Transport Layer
- Application Layer
- Sockets
- Wireshark



Course Methodology

Most lessons follow the format below:

- Introduce a network stack layer
- Discuss specific protocols within the layer, including:

Ethernet	UDP	ARP
IPv4	DHCP	IPv6
ICMP	DNS	ICMPv6
TCP	HTTP/HTTPS	DHCPv6

- Apply the concepts in a lab
- This test is LAB HEAVY

ADVANCED CYBER TRAINING PROGRAM

Learning Objectives

Given a workstation, device, and/or technical documentation, the student will be able to:

- Use common protocol specifications to analyze, interpret, and construct network traffic for the following Open Systems Interconnection (OSI) reference model layers:
  - Data Link (Layer 2)
  - Network (Layer 3)
  - Transport (Layer 4)
  - Application (Layer 7)
- Use Sockets to create simple network tools

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

But First!

Lets cover some important concepts

Linux VM

- Networking lends itself well to using two machines for testing
- You have a Linux VM in virtualbox titled Fedora 30 to use in tandem with your host machine.
    - This will also be used for the kernel class, so take a snapshot of a fresh state before you start using it

Bits and Bytes and Chocolate Cake

- 1 Bit = Single value, either 1 or 0 (True or False)
- 1 Nibble/nybble/nyble = 4 Bits = ½ Byte = 1 hex digit
- 1 Octet = 8 Bits
- 1 Byte = smallest addressable unit of memory
  - Most computers today define 8 Bits to be a 1 Byte
  - Older systems have used 4, 6, and 7 Bits to define a Byte
  - IEEE 1541 defines “A byte is a set of adjacent bits operated on as a group”

Endianess

- Big Endian
  - Fields are left to right with most significant octet on the left and least significant octet on the right
  - In memory the most significant octet is stored in the lowest address
  - Network Byte order - Defined by RFC 1700, later superseded by RFC 3232
  - Processors: IBM and Motorola
- Little Endian
  - Fields are right to left with least significant octet on the left
  - In memory the least significant octet is stored in the lowest address
  - Processors: Intel®

Endianess

- Hexadecimal number: 0xAABBCCDD

Big Endian

ADDRESS	0x00010000	0x00010001	0x00010002	0x00010003
Data	0xAA	0xBB	0xCC	0xDD

Little Endian

ADDRESS	0x00010000	0x00010001	0x00010002	0x00010003
Data	0xDD	0xCC	0xBB	0xAA

UDEV device manager

- udev
  - Linux user space device manager
  - Provides persistent device naming regardless of the order in which the devices are connected to the system
  - Network interfaces renamed
    - No more “eth0” - now “enp2s0” or similar
  - http://www.freedesktop.org/wiki/Software/systemd/PredictableNetworkInterfaceNames/
  - Let’s break it down: [en][p2[s0]]
    - en – Ethernet (CAT cable)
    - wl – wlan (WiFi card)
    - ww - wwan (GSM, WiMAX, Cellular Digital Packet Data)
    - p<bus>s<slot> (u<slot> for USB based devices)
    - Verify with ls[pci|usb|pcmia]

Device Naming Convention

- enp2s0 is a better scheme
  - Stable interface names across reboots
  - Stable interface names even when hardware is added or removed, i.e. no re-enumeration takes place
  - Stable interface names when kernels or drivers are updated/changed
  - Stable interface names even if you have to replace broken Ethernet cards by new ones
  - The interface names are fully predictable, i.e. just by looking at lspci you can figure out what the interface is going to be called

Linux Helper: Linux Networking Commands

- Query network interfaces

```
$ ifconfig -a
```

- Query routing table

```
$ route [-FC]
```

- Manipulate the system ARP cache

```
$ arp [-n]
```

- show / manipulate routing, devices, policy routing and tunnels

```
$ ip [addr | route | neigh]
```

- Control network interface

```
$ sudo service network-manager [start|stop|restart]
$ sudo systemctl restart NetworkManager.service
```

iproute2 vs net-tools

- Currently there is a transition happening in the network utility world. Net-tools has been deprecated and is being replaced iproute2.
- This is why on many newer systems, commands like ifconfig and netstat may not be available by default. You can always install the net-tools package
- The table below shows a few commands and compares between the two

Net-tools	Iproute2	Details
Ifconfig	ip addr	Display address information
Netstat	ss	Display socket information
Route	ip route	Display routing information
Arp	Ip neigh	Display arp and neighbor table
Brctl	bridge	Control bridging

Physical Layer References

- For two systems to communicate they must be connected to the same type of physical medium
- http://www.inetdaemon.com/tutorials/basic_concepts/communication/transmission/transmission_media.shtml
- http://www.inetdaemon.com/tutorials/networking/lan/ethernet/index.shtml
- http://www.inetdaemon.com/tutorials/networking/lan/fddi/index.shtml

Link Layer References

- The Ethernet: A Local Area Network, Data Link Layer and Physical Layer Specifications, Version 2
- **IEEE 802.3** – LAN/MAN CSMA/CD (Ethernet) Access Method and Physical Layer Specifications
- IEEE 802.1D – LAN/MAN Media Access Control (MAC) Bridges
- RFC 1122 – Requirements for Internet Hosts -- Communication layers
- RFC 1123 – Requirements for Internet Hosts -- Application and Support
- RFC 893 – Trailer Encapsulations
- **RFC 826** – An Ethernet Address Resolution Protocol
- RFC 894 – A Standard for the Transmission of IP Datagrams over Ethernet Networks
- RFC 1042 – A Standard for the Transmission of IP Datagrams over IEEE 802 Networks
- RFC 2740 – OSPF for IPv6

Network Layer References

- **RFC 791** – Internet Protocol
- **RFC 8200** – IPv6 Internet Protocol
- RFC 2675 – IPv6 Jumbograms
- **RFC 4443** – ICMPv6
- RFC 1700 – Assigned Numbers
- RFC 919 – Broadcasting Internet Datagrams
- RFC 1812 – Requirements for IP Version 4 Routers
- RFC 1122 – Requirements for Internet Hosts – Communication Layers
- RFC 917 – Internet subnets
- RFC 3439 – Some Internet Architectural Guidelines and Philosophy
- RFC 950 – Internet Standard Subnetting Procedure
- RFC 4632 – Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan
- **RFC 792** – Internet Control Message Protocol
- IEEE EUI-64 – http://standards.ieee.org/develop/regauth/tut/eui64.pdf

Transport Layer References

- **RFC 768 – User Datagram Protocol**
- **RFC 9293 – Transmission Control Protocol**
- RFC 879 (7805) – TCP maximum segment size and related topics
- RFC 1122 – Requirements for Internet Hosts – Communication Layers
- RFC 2018 – TCP Selective Acknowledgement Options
- RFC 7323 – TCP Extensions for High Performance
- RFC 7540 – Hypertext Transfer Protocol Version 2 (HTTP/2)
- ITU-T Recommendation X.224: Information technology - Open Systems Interconnection - Protocol for providing the connection-mode transport service
- ITU-T Recommendation X.234: Information technology - Protocol for providing the OSI connectionless-mode transport service
- Sockets
  - POSIX 1003.1-2001 (Berkeley Sockets)
  - MSDN (WinSock)
  - Scapy (www.secdev.org/projects/scapy/)

Application Layer References

- RFC-951 – Bootstrap Protocol (BOOTP)
- RFC-2131 – Dynamic Host Configuration Protocol (DHCP)
- RFC-8415 – Dynamic Host Configuration Protocol for IPv6 (DHCPv6)
- RFC-1945 – Hypertext Transfer Protocol – HTTP/1.0
- RFC-2616 – Hypertext Transfer Protocol – HTTP/1.1
- RFC-7540 – Hypertext Transfer Protocol – HTTP/2.0
- RFC-9114 – Proposed Standard Hypertext Transfer Protocol – HTTP3 and QUIC
- RFC-8446 – The TLS Protocol Version 1.3
- RFC-2818 – HTTP Over TLS
- RFC-5280 – Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
- RFC-1034 – Domain Names – Concepts and Facilities
- RFC-1035 – Domain Names – Implementation and Specification
- RFC-1001 – Protocol standard for a NetBIOS service on a TCP/UDP transport: Concepts and methods

Application Layer References *(continued)*

- RFC-1002 – Protocol standard for a NetBIOS service on a TCP/UDP transport: Detailed specifications
- RFC-1459 – Internet Relay Chat Protocol (IRC)
- [MS-CIFS]: Common Internet File System (CIFS) Protocol Specification
- [MS-SMB]: Server Message Block (SMB) Protocol Specification

Additional References

- Transmisssion Error (corruption) Detection and Correction
  - http://www.cs.gmu.edu/~huangyih/656/error.pdf

HOW TO: Win at Networks!

- Read the RFC’s
  - Can’t be stressed enough
- Wireshark!
  - Mimic the packet type you are trying to send (usually involves issuing a command on the command line) and compare it with what you are sending.
  - Example: Trying to manually send a TCP packet? Browse to the test server – HTTP runs on top of TCP. View in Wireshark a correct example of TCP!
  - Display filters vs Capture filters
- Man pages of commands (ex: man ping)
- Take a guess and try it! Worst thing that could happen is somebody has to reboot.
- Make sure your using byte strings because it’s Python 3
- Ask

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

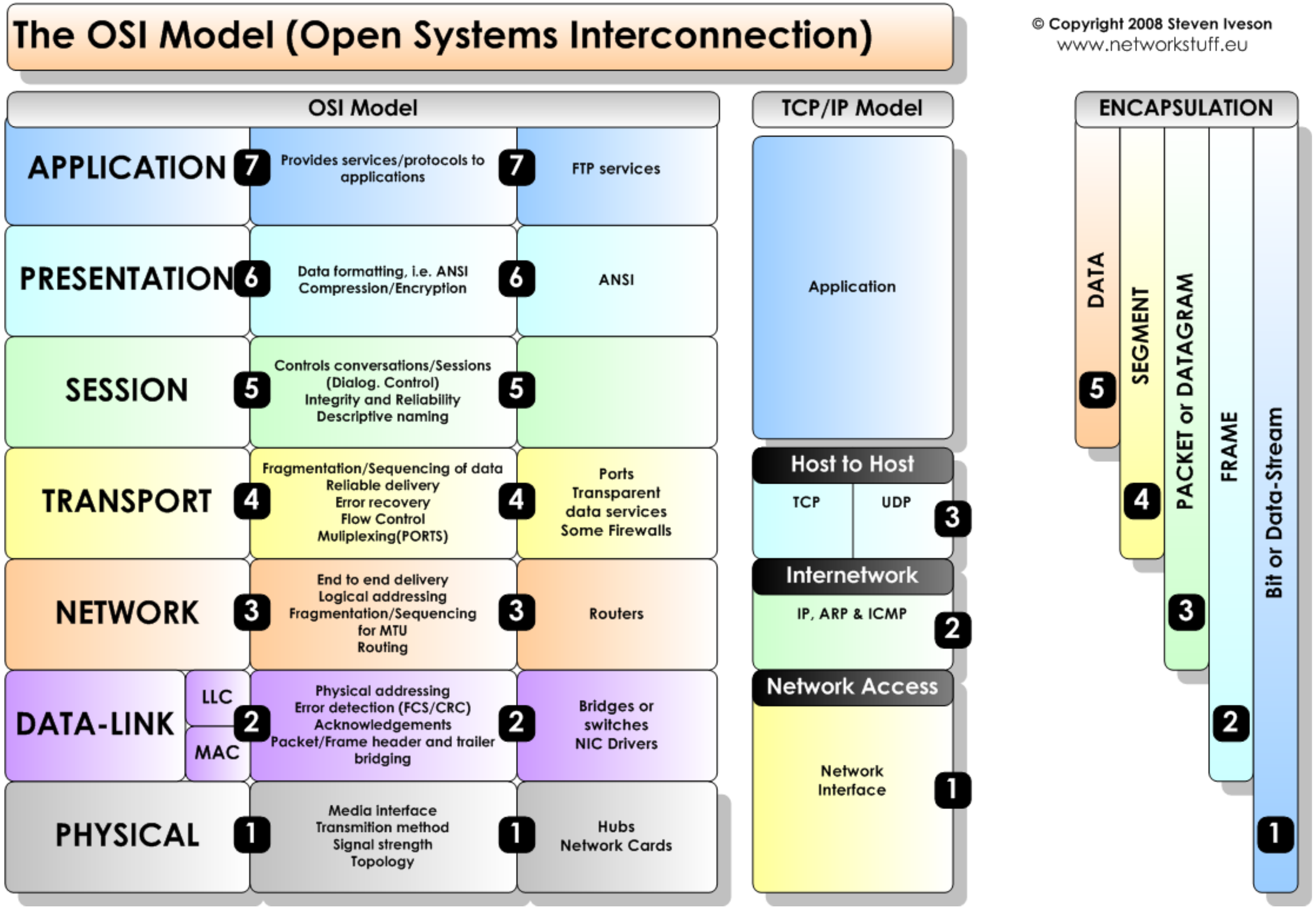
Background

A Network Overview

Network models

- Open Systems Interconnection (OSI) model
  - 7 Layers
  - Defined by ISO/IEC 7498-1
  - **Reference model**
    - Defines strict hierarchy of layers
    - Focuses on providing a reliable data transfer service
- TCP/IP model
  - 4 Layers
  - Maintained by the Internet Engineering Task Force (IETF)
    - RFC 1122 - Host Requirements
  - Funded by Defense Advanced Research Projects Agency (DARPA)
    - Also known as the DOD model
  - **Implementation model**
    - Applications use only what is required
    - Layers only focus on the layer below

Open Systems Interconnection (OSI) Model



The OSI Model (Open Systems Interconnection)

© Copyright 2008 Steven Iveson
www.networkstuff.eu

OSI Model				
APPLICATION	7	Provides services/protocols to applications	7	FTP services
PRESENTATION	6	Data formatting, i.e. ANSI Compression/Encryption	6	ANSI
SESSION	5	Controls conversations/Sessions (Dialog. Control) Integrity and Reliability Descriptive naming	5	
TRANSPORT	4	Fragmentation/Sequencing of data Reliable delivery Error recovery Flow Control Muliplexing(PORTS)	4	Ports Transparent data services Some Firewalls
NETWORK	3	End to end delivery Logical addressing Fragmentation/Sequencing for MTU Routing	3	Routers
DATA-LINK (LLC / MAC)	2	Physical addressing Error detection (FCS/CRC) Acknowledgements Packet/Frame header and trailer bridging	2	Bridges or switches NIC Drivers
PHYSICAL	1	Media interface Transmition method Signal strength Topology	1	Hubs Network Cards

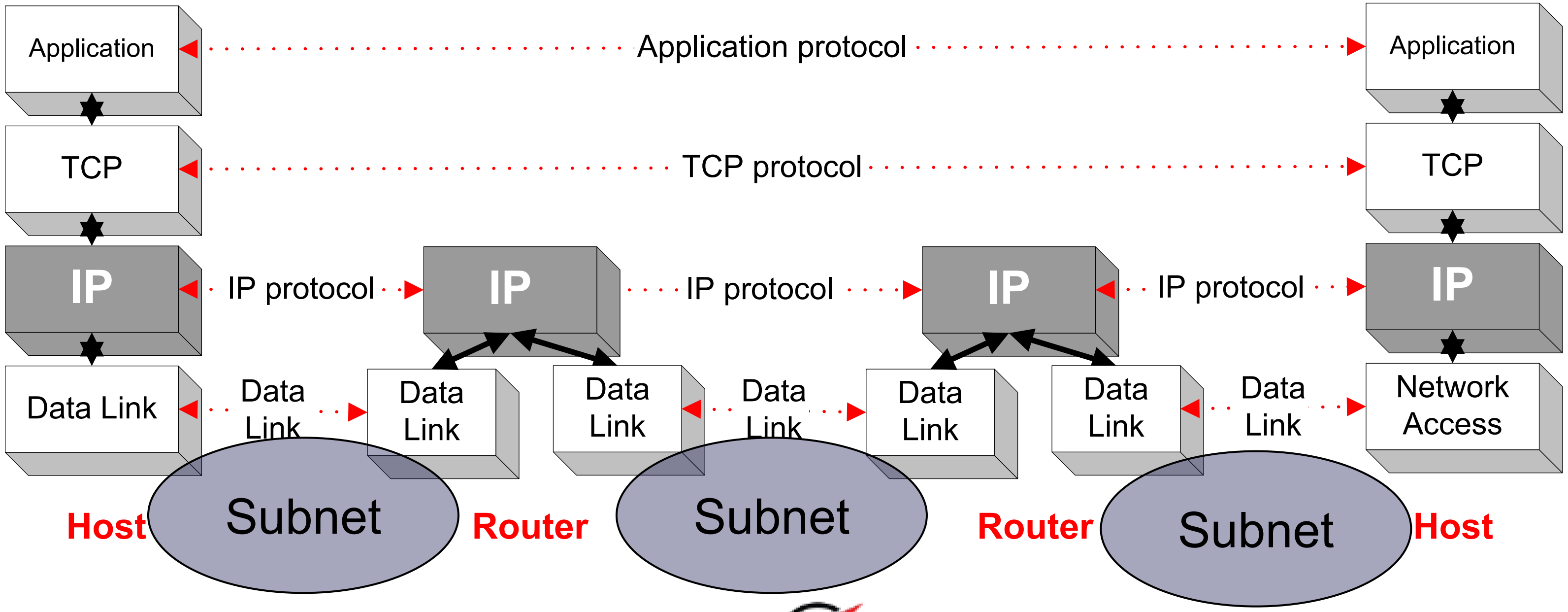
TCP/IP Model	
Application	
Host to Host: TCP, UDP	3
Internetwork: IP, ARP & ICMP	2
Network Access: Network Interface	1

ENCAPSULATION	
DATA	5
SEGMENT	4
PACKET or DATAGRAM	3
FRAME	2
Bit or Data-Stream	1

ManTech

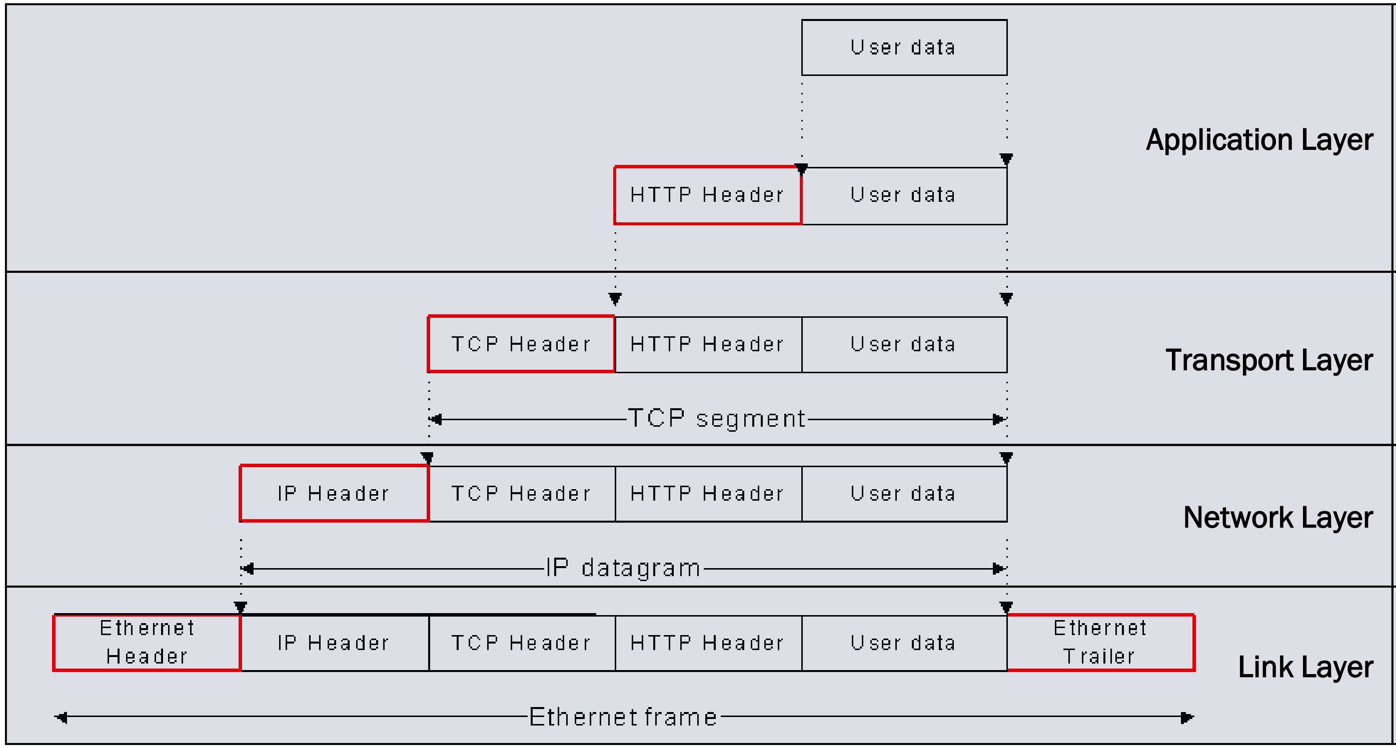
Big Picture

- Link Layer (Ethernet) allows you to talk in a subnet
- Network Layer (IP) allows you to talk across multiple subnets
- Transport Layer (TCP/UDP) separates network conversations between hosts
- Application Layer processes the data at the host



TCP/IP Model (or DOD Model)

We can’t cover all of these



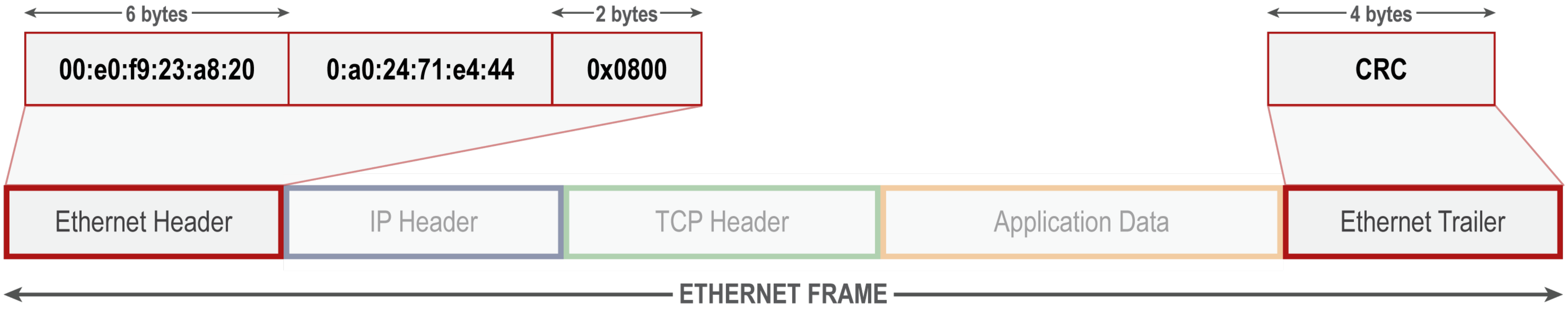
Application Layer	BGP, DHCP, DNS, FTP, GTP, HTTP, IMAP, IRC, NTP, POP, RIP, RPC, RTP, RTSP, SDP, SIP, SMTP, SNMP, SSH, STUN, Telnet, TLS/SSL…etc
Transport Layer	GRE, OSPF, TCP, UDP, WTLS…etc
Network Layer	IP, ICMP, IGMP, IPsec..etc
Link Layer	MAC (Ethernet), ARP, RARP, OSPF, Device Drivers…etc

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

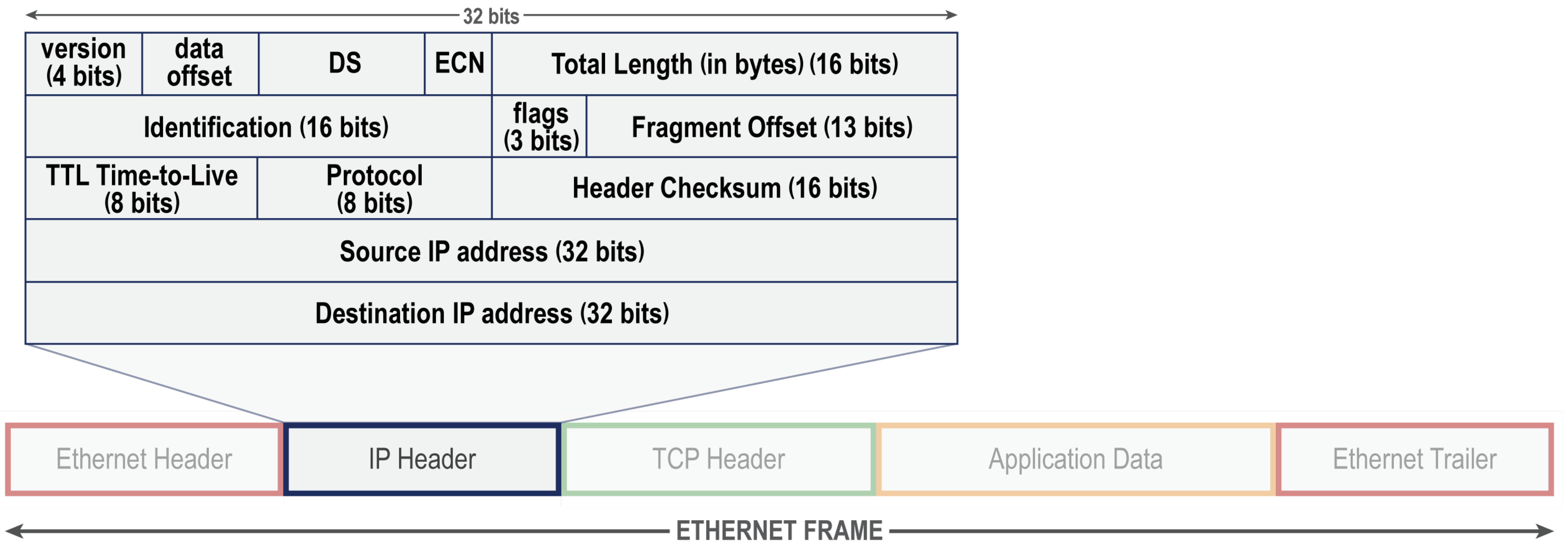
Link Layer

Ethernet Data Fields



IPv4 Data Fields

RFCs:
791

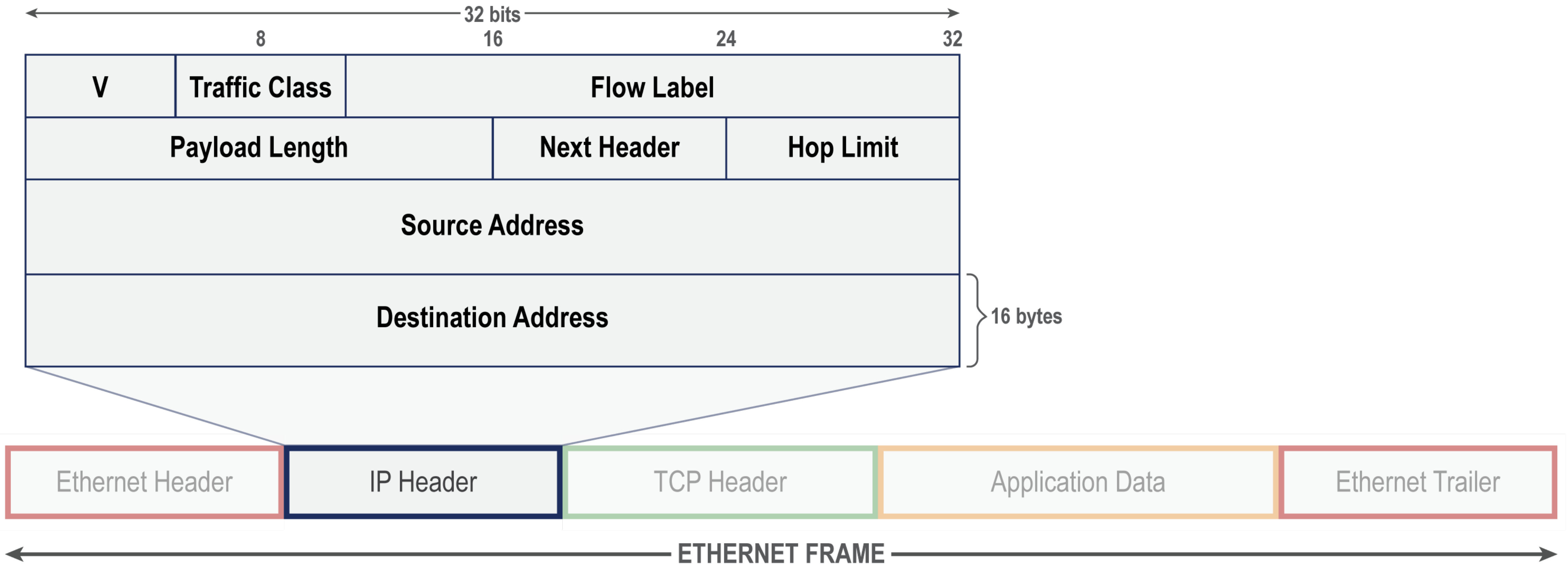


32 bits

<table>
<tr><td>version
(4 bits)</td><td>data
offset</td><td>DS</td><td>ECN</td><td colspan="2">Total Length (in bytes) (16 bits)</td></tr>
<tr><td colspan="4">Identification (16 bits)</td><td>flags
(3 bits)</td><td>Fragment Offset (13 bits)</td></tr>
<tr><td colspan="2">TTL Time-to-Live
(8 bits)</td><td colspan="2">Protocol
(8 bits)</td><td colspan="2">Header Checksum (16 bits)</td></tr>
<tr><td colspan="6">Source IP address (32 bits)</td></tr>
<tr><td colspan="6">Destination IP address (32 bits)</td></tr>
</table>

IPv6 Data Fields

RFCs:
8200



32 bits

8 16 24 32

V | Traffic Class | Flow Label

Payload Length | Next Header | Hop Limit

Source Address

Destination Address

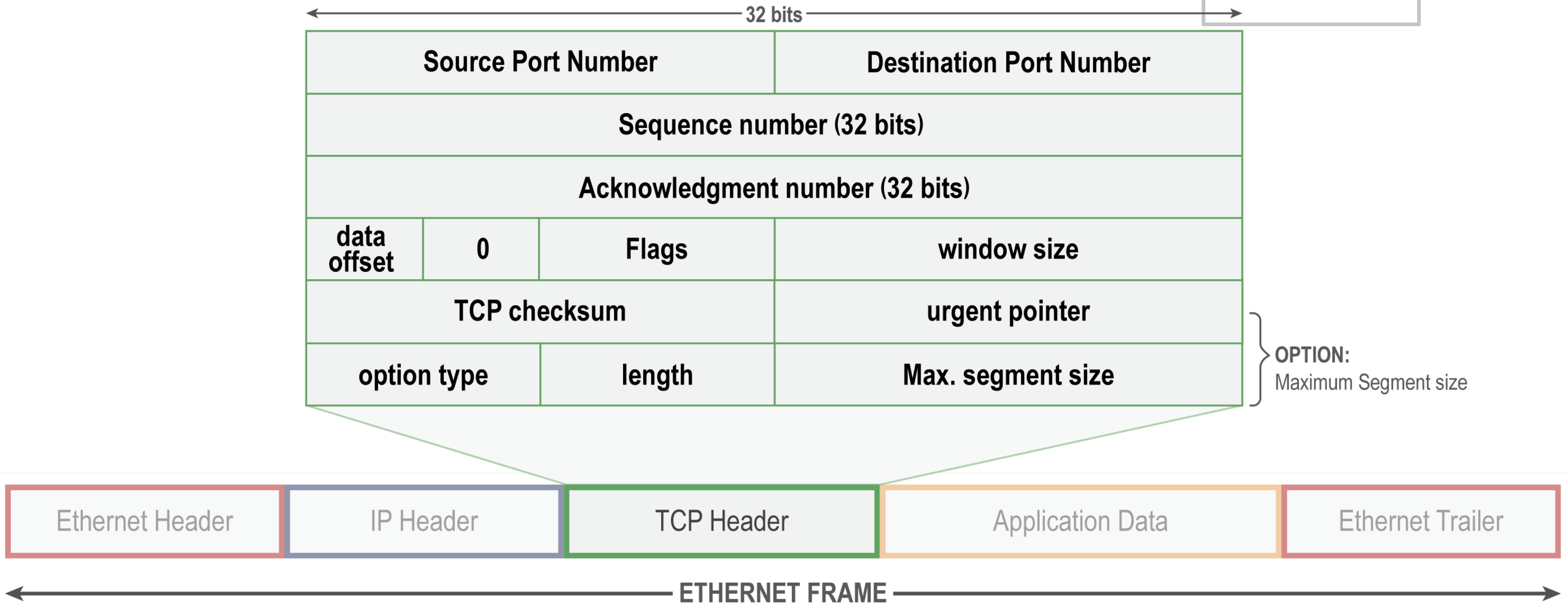
16 bytes

Ethernet Header | IP Header | TCP Header | Application Data | Ethernet Trailer

ETHERNET FRAME

TCP Data Fields

RFCs:
9293



32 bits

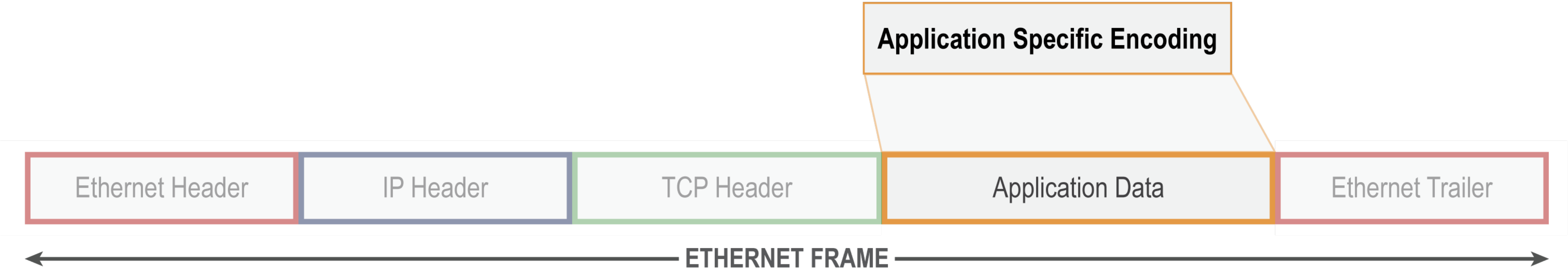
Source Port Number
Destination Port Number
Sequence number (32 bits)
Acknowledgment number (32 bits)
data offset
0
Flags
window size
TCP checksum
urgent pointer
option type
length
Max. segment size

OPTION:
Maximum Segment size

Ethernet Header
IP Header
TCP Header
Application Data
Ethernet Trailer

ETHERNET FRAME

Application Data



Application Specific Encoding

Ethernet Header

IP Header

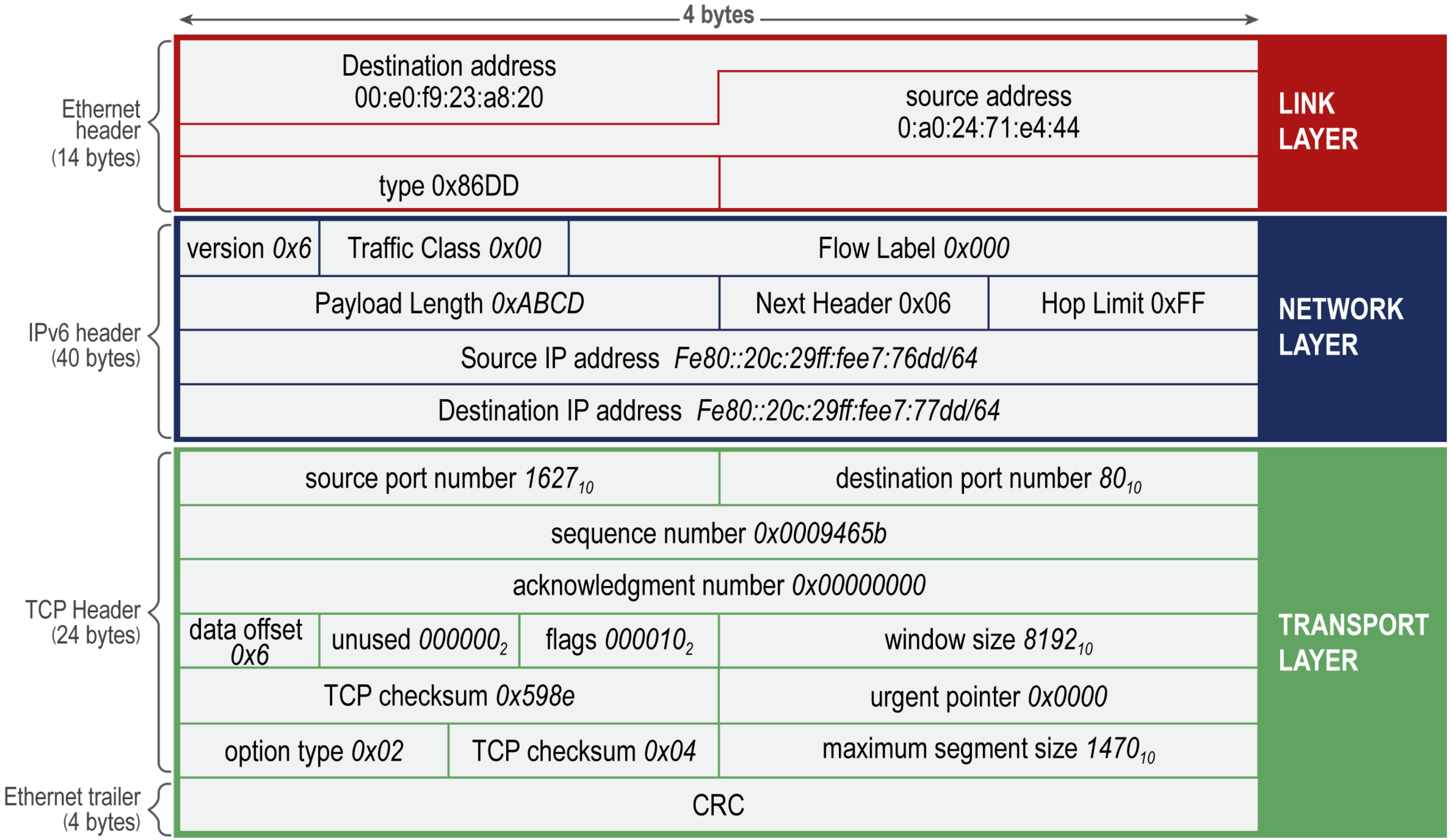
TCP Header

Application Data

Ethernet Trailer

ETHERNET FRAME

Typical Frame Breakdown



Ethernet header (14 bytes)	Destination address 00:e0:f9:23:a8:20		source address 0:a0:24:71:e4:44		LINK LAYER
	type 0x86DD				
IPv6 header (40 bytes)	version *0x6*	Traffic Class *0x00*	Flow Label *0x000*		NETWORK LAYER
	Payload Length *0xABCD*		Next Header 0x06	Hop Limit 0xFF	
	Source IP address *Fe80::20c:29ff:fee7:76dd/64*				
	Destination IP address *Fe80::20c:29ff:fee7:77dd/64*				
TCP Header (24 bytes)	source port number 1627_{10}		destination port number 80_{10}		TRANSPORT LAYER
	sequence number *0x0009465b*				
	acknowledgment number *0x00000000*				
	data offset *0x6*	unused 000000_2	flags 000010_2	window size 8192_{10}	
	TCP checksum *0x598e*		urgent pointer *0x0000*		
	option type *0x02*	TCP checksum *0x04*	maximum segment size 1470_{10}		
Ethernet trailer (4 bytes)	CRC				

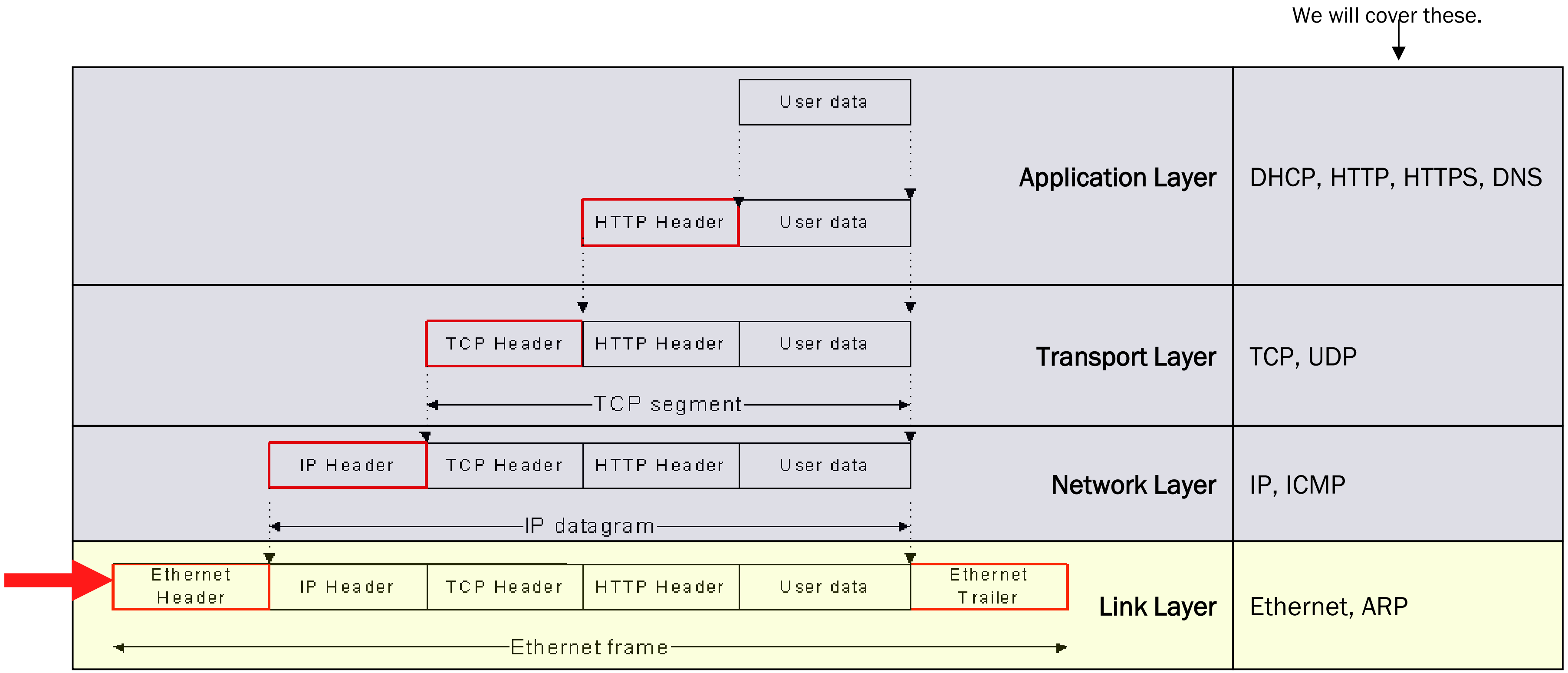
Objectives

Given a workstation, device, and/or technical documentation, the student will be able to:

- Learning Objective
  - Use common Link Layer protocol specifications to analyze, interpret, and construct network traffic
- Enabling Objectives
  - Describe the display frame with breakdown of all fields
  - Describe collision and broadcast domains
  - Construct raw Ethernet frames and ARP requests

Link Layer

- A group of methods, protocols, and specifications that is closest to the physical network components.



Ethernet Frame – Example

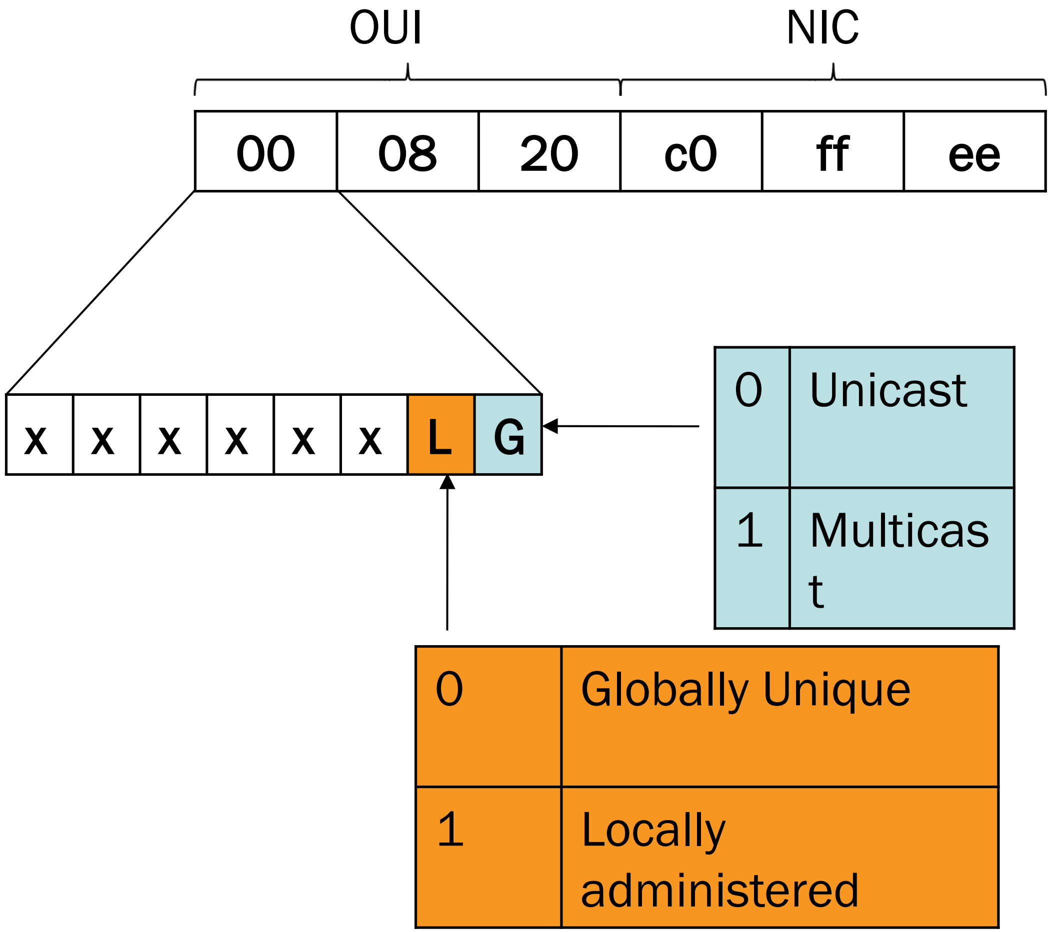
destination address	source address	type	data	CRC
6	6	2	46-1500	4

0800	IP datagram
2	46-1500

0806	ARP request/reply	PAD
2	28	18

Ethernet Addresses

Source:
IEEE 802.3 Specification



OUI			NIC		
00	08	20	c0	ff	ee

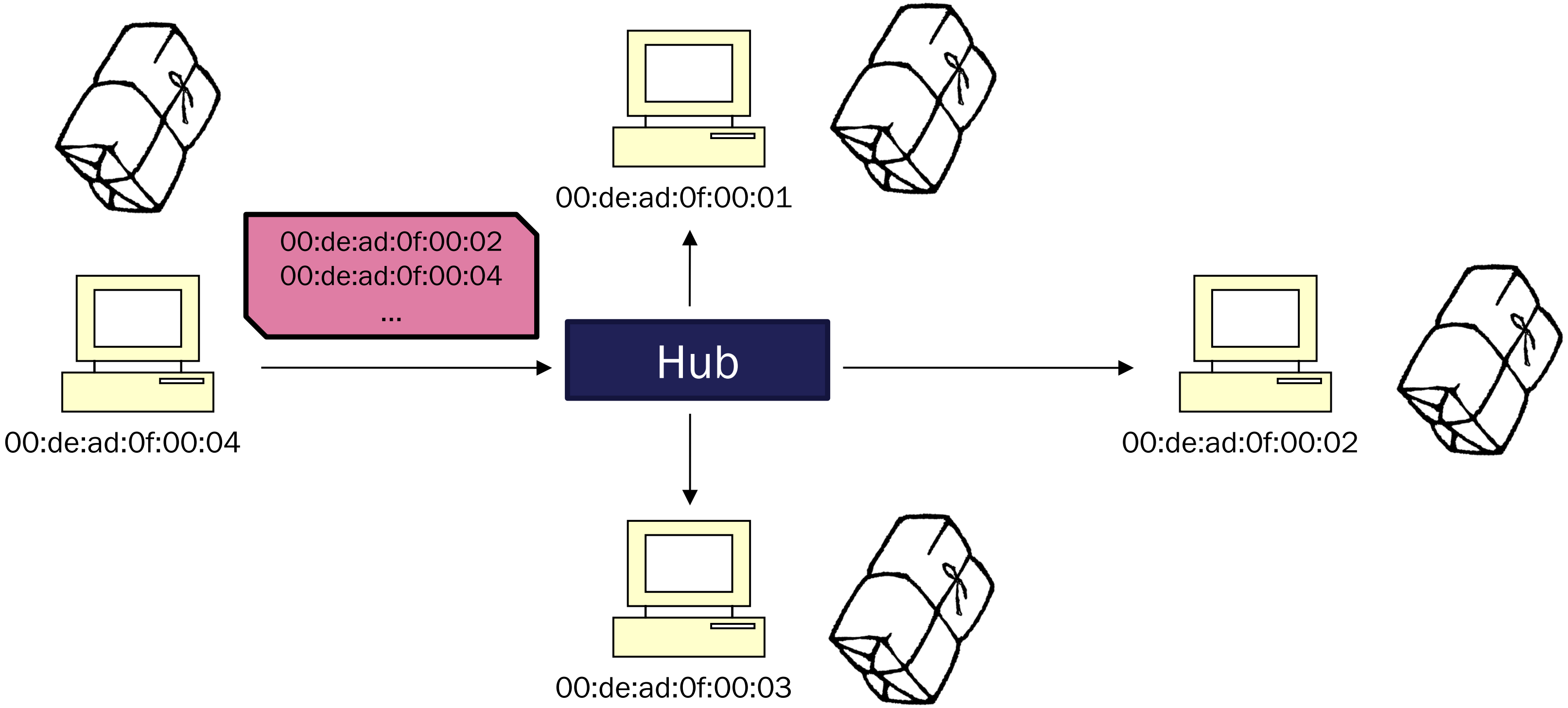
x	x	x	x	x	x	L	G

0	Unicast
1	Multicast

0	Globally Unique
1	Locally administered

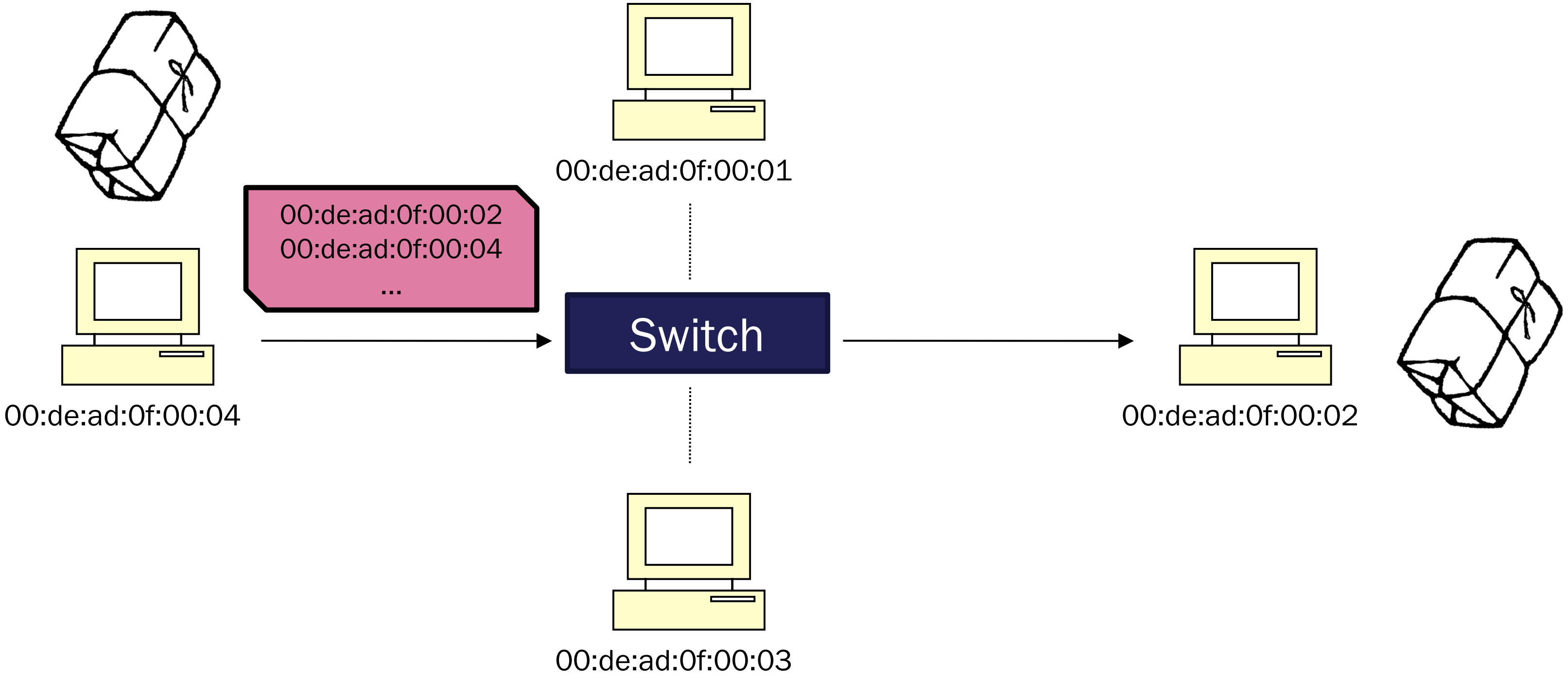
Hubs

- All frames forwarded to all physical ports
- Single Collision Domain, Single Broadcast Domain



Switches

- Ethernet Address/Physical Port associations “learned”
- Multiple Collision Domains, Single Broadcast Domain



Forwarding Table

- Lookup physical port by Ethernet address
  - Single physical port may have a number of associated Ethernet addresses
  - The switch uses the source mac address of a frame on a specific port to populate the table
- Implementation variation
  - RAM/CAM
  - Max Entries
  - Aging

Table Example:

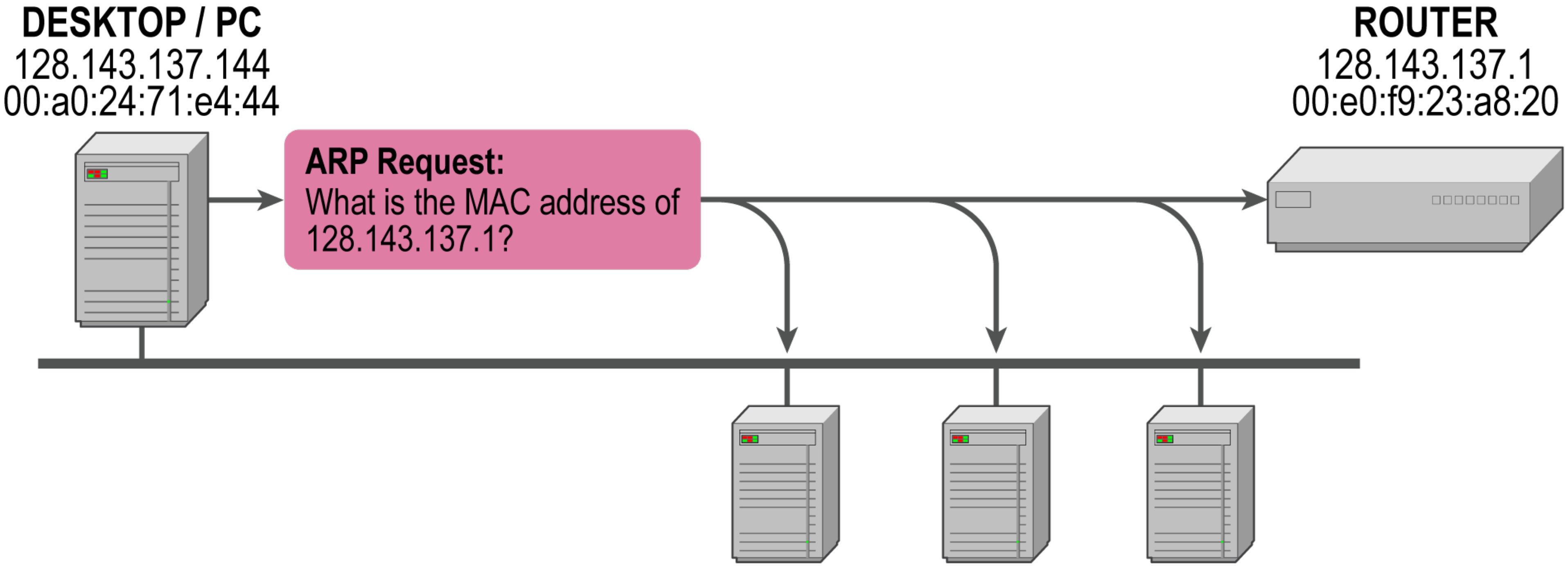
00:de:ad:0f:00:01	2
00:de:ad:0f:00:02	1
00:de:ad:0f:00:03	4
00:de:ad:0f:00:04	3
00:de:ad:0f:00:05	3

Address Resolution Protocol (ARP)

- ARP Request

  Argon broadcasts an ARP request to all stations on the network:

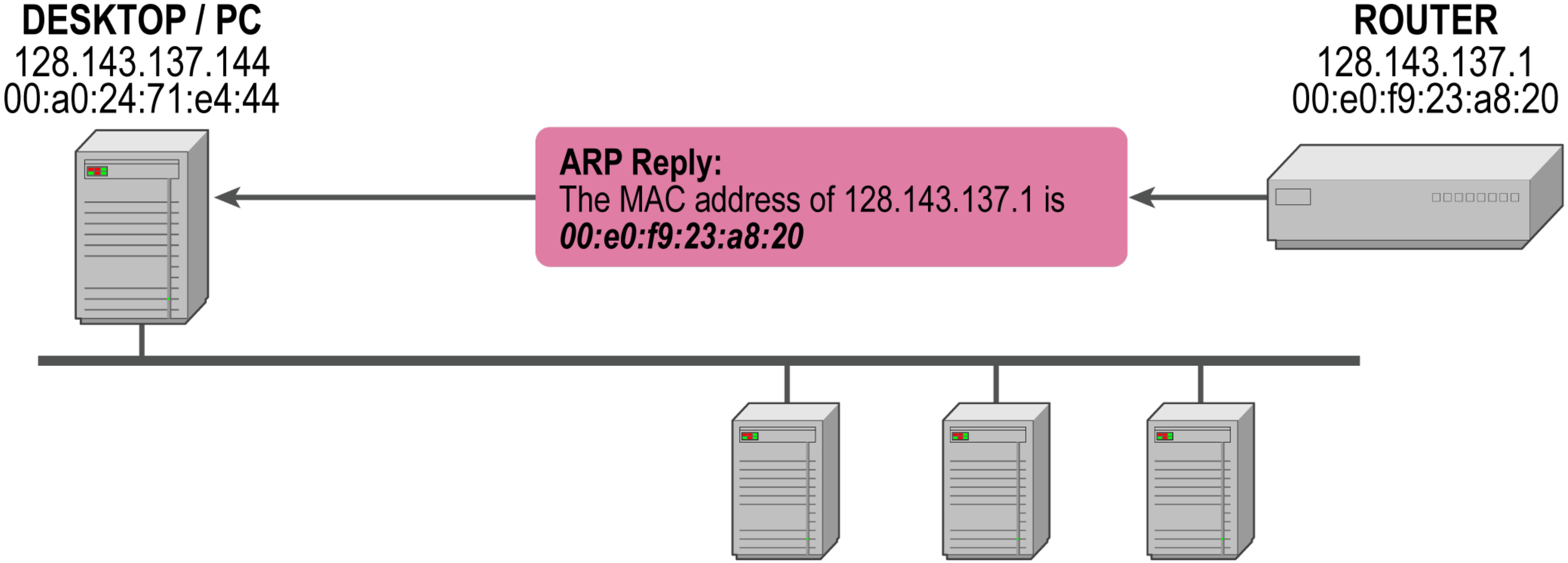
  “What is the hardware address of Router?”



ARP Request:
What is the MAC address of 128.143.137.1?

Address Resolution Protocol *(continued)*

- **ARP Reply:**
  Router responds with an ARP Reply which contains the hardware address



DESKTOP / PC
128.143.137.144
00:a0:24:71:e4:44

ROUTER
128.143.137.1
00:e0:f9:23:a8:20

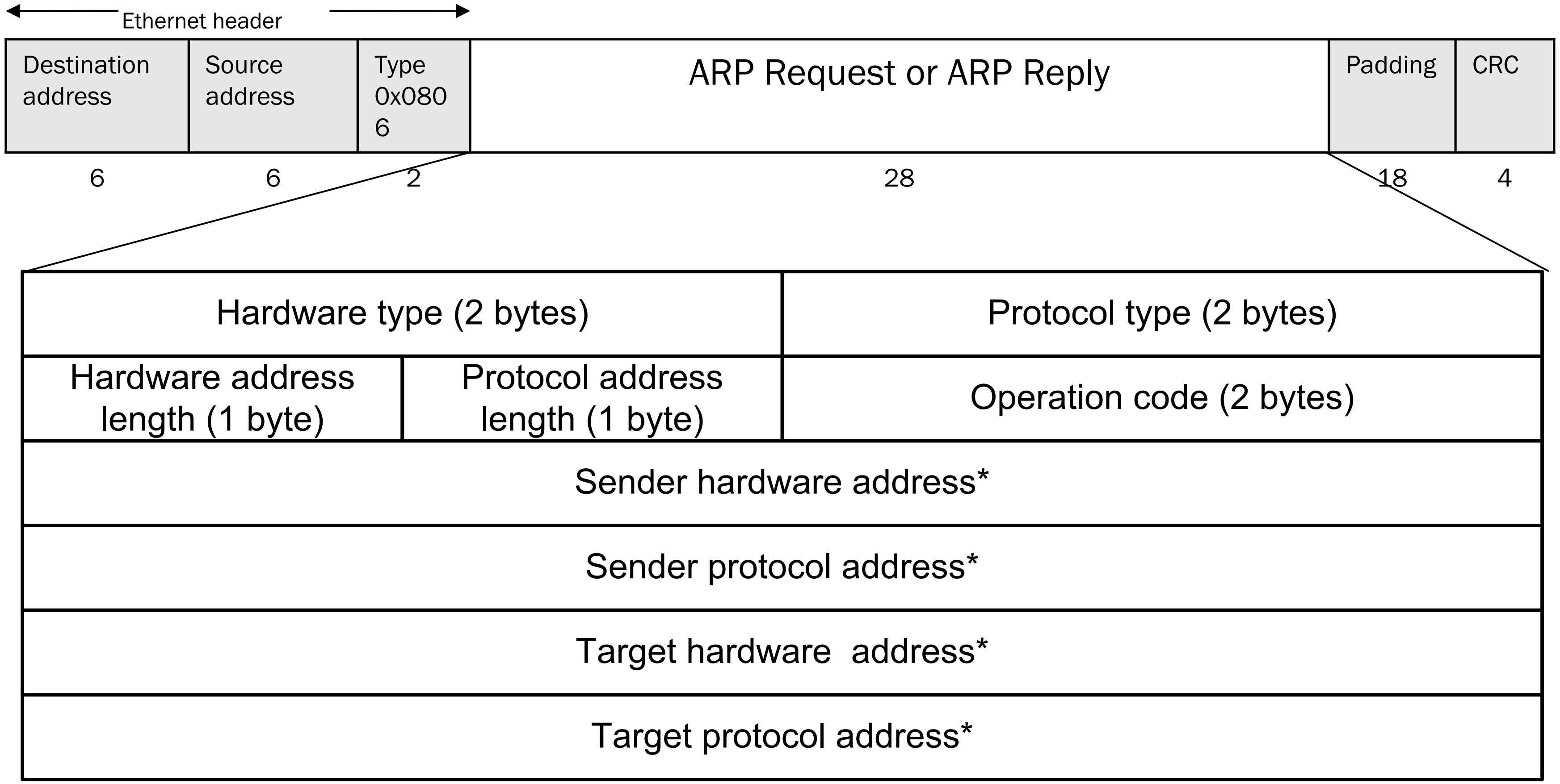
ARP Reply:
The MAC address of 128.143.137.1 is
00:e0:f9:23:a8:20

LINUX
CNO
Programming

ADVANCED CYBER TRAINING PROGRAM

ARP Header

ARP Packet Format



Ethernet header

Destination address	Source address	Type 0x0806	ARP Request or ARP Reply	Padding	CRC
6	6	2	28	18	4

<table>
<tr><td colspan="2">Hardware type (2 bytes)</td><td colspan="2">Protocol type (2 bytes)</td></tr>
<tr><td>Hardware address length (1 byte)</td><td>Protocol address length (1 byte)</td><td colspan="2">Operation code (2 bytes)</td></tr>
<tr><td colspan="4">Sender hardware address*</td></tr>
<tr><td colspan="4">Sender protocol address*</td></tr>
<tr><td colspan="4">Target hardware address*</td></tr>
<tr><td colspan="4">Target protocol address*</td></tr>
</table>

* Note: The length of the address fields is determined by the corresponding address length fields

ARP Hardware Types (subset)

TYPE ID	DESCRIPTION	HW ADDR LENGTH
1	Ethernet	6 bytes
…	…	…
18	Fibre Channel	6 bytes
…	…	…
24	IEEE1394.1995	16 bytes
…	…	…
32	Infiniband	20 bytes

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

Lab 1

Link Layer – Ethernet and Introduction of Wireshark

ManTech

Overview

- **TASK 1**: Pings
- **TASK 2:** Caches for ARP and IP
- **TASK 3:** ARP Requests
- **TASK 4:** MITM with ARP

Lab Intro: Clarifications

- Network stack shortcuts/optimizations
  - Sending a packet from YOUR port X to YOUR port Y, does it have to go on the wire?
  - If it doesn’t go on the wire, will Wireshark see it?
  - Get to know your neighbors! You’ll be sitting next to them for the next 3 months
- Bridged vs. NAT

Lab Intro

- Experiment as time permits, you will learn a lot more
- Don’t be afraid of breaking something! Don’t be afraid of breaking a lot either – it’s just software
- Reboot if need be
  - Windows: Control Panel\Network and Internet\Network Connections, Disable then Enable
  - Linux: /etc/init.d/networking restart
    - Distro specific
  - Any OS: Reboot the system
- WIRESHARK IS YOUR FRIEND!!!!
  - Use it to compare what **is** being sent on the wire vs. what you **think** you are sending out

Handy Wireshark Display Filters

- ip.host == 192.168.1.37
    - Display only traffic to or from IP addr 192.168.1.37
- ipv6.host == fe80::20c:29ff:fe39:5
    - Display only traffic to or from IPv6 addr fe80::20c:29ff:fe39:5
- ipv6.host >= fe80::1 && ipv6.host <= fe80:ffff:ffff:ffff:ffff:ffff:ffff:ffff
    - Display only traffic to/from the fe80::1 to fe80:ffff:ffff:ffff:ffff:ffff:ffff:ffff
- ipv6.src == fe80::20c:29ff:fe39:5b8e
    - Display only traffic with the source being fe80::20c:29ff:fe39:5b8e
- ipv6.dst == fe80::20c:29ff:fe39:5b8e
    - Display only traffic with the destination being fe80::20c:29ff:fe39:5b8e
- tcp.port == 80 && ip.version == 6
    - Display only traffic on port 80
- ip.host == 172.16.0.2 && tcp.port == 80
    - Display any traffic involving google.com that isn't port 80 traffic
- ip.host == 172.16.0.2 && !(tcp.port == 80 || tcp.port == 25)
    - Display any traffic involving google.com that isn't port 80 or 25
- ipv6
    - Display only IPv6 traffic. Other common filters for protocols include ip
    - http, icmp, icmpv6, and dns.
- eth.addr == 00:11:22:33:44:55
    - Display traffic with a src or dst address of 00:11:22:33:44:55

TASK 0: Wireshark

- Demo:
  1. Viewing available interfaces
  2. Starting a capture
  3. Changing the layout
  4. Enable automatically scrolling to last packet
  5. Viewing packets
  6. Restarting a capture
  7. Show filters and how to view all filters

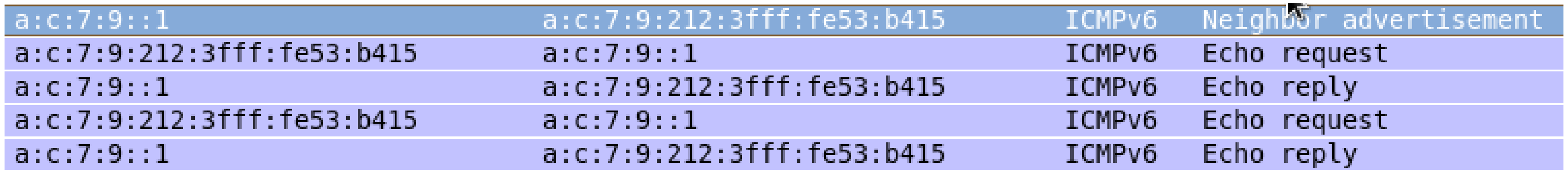
TASK 1: Ping and Ping 6

- Start Wireshark
- Apply filter “icmp”
- Run `ping -c 1 <ipv4>`
- Apply filter “icmpv6”
- Run `ping6 -c 1 <ipv6>`
    - Is this `ping6` command Linux or Windows? And what is the other one’s equivalent?
- What was transmitted on the network?
- Did both endpoints do the same things?
- What was different between ping and ping6?
- Remove any filters and look at other traffic surrounding your pings

Ping6 Example

```
[plussign@panda ~]# ping6 -c 1 a:c:7:9::1
PING a:c:7:9::1(a:c:7:9::1) 56 data bytes
64 bytes from a:c:7:9::1: icmp_seq=1 ttl=64 time=1.24 ms

--- a:c:7:9::1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time
1ms
rtt min/avg/max/mdev = 1.248/1.248/1.248/0.000 ms
```



```
a:c:7:9::1                  a:c:7:9:212:3fff:fe53:b415   ICMPv6  Neighbor advertisement
a:c:7:9:212:3fff:fe53:b415  a:c:7:9::1                   ICMPv6  Echo request
a:c:7:9::1                  a:c:7:9:212:3fff:fe53:b415   ICMPv6  Echo reply
a:c:7:9:212:3fff:fe53:b415  a:c:7:9::1                   ICMPv6  Echo request
a:c:7:9::1                  a:c:7:9:212:3fff:fe53:b415   ICMPv6  Echo reply
```

TASK 2: ARP & IP Caches

```
[plussign@panda ~]$ ip neigh show
172.17.0.20 dev eth0 lladdr 00:0c:29:fe:ce:7c REACHABLE
172.17.0.100 dev eth0 lladdr 00:15:c5:e0:d4:a4 REACHABLE
[plussign@panda ~]# ip -6 neigh show
a:c:7:9::1 dev eth0 lladdr 00:0c:29:39:5b:8e router REACHABLE
```

- View the cache, connect to a new host, and view the cache again
- What do each of these values mean?
- What's in YOUR caches?

TASK 3: DIY ARP Request

- Given slides and capture, construct the Ethernet header of an ARP Request frame as a python string:

```
>>> dst = b“\xff” * 6     # Broadcast
>>> src = b“\x00\xde\xad\xf0\x00\x01” # YOUR MAC
>>> typ = b“\x08\x06”  # ARP Type
>>> etherhdr = dst + src + typ
```

TASK 3: DIY ARP Request *(continued)*

- Given slides and capture, construct the ARP Request portion of the frame as a python string:

```
>>> arpreq = b“\x00\x01”          # Hw Type
>>> arpreq += b“\x08\x00”         # Proto Type
>>> arpreq += b“\x06”    # Hw Addr Len
>>> arpreq += b“\x04”    # Proto Addr Len
>>> arpreq += b“\x00\x01”         # Op Code (Req)
>>> arpreq += b“\x00\xde\xad\xf0\x00\x01”      # YOUR MAC (SRC)
>>> arpreq += b“\xc0\xa8\x00\x80”        # YOUR IP (SRC)
>>> arpreq += b“\x00” * 6         # Zeroed on Request
>>> arpreq += b“\xc0\xa8\x00\x42”        # THEIR IP(DST)
```

TASK 3: DIY ARP Request *(continued)*

- Make sure your sniffer is running
  - Apply the wireshark filter: *arp && eth.addr == <your mac address>*
- Using provided helper functions, send your frame

```
>>> import cno_net_helper as cno
>>> cno.rawsend(etherhdr + arpreq, dev="ens33")
Note: use ip addr to grab your interface.
```

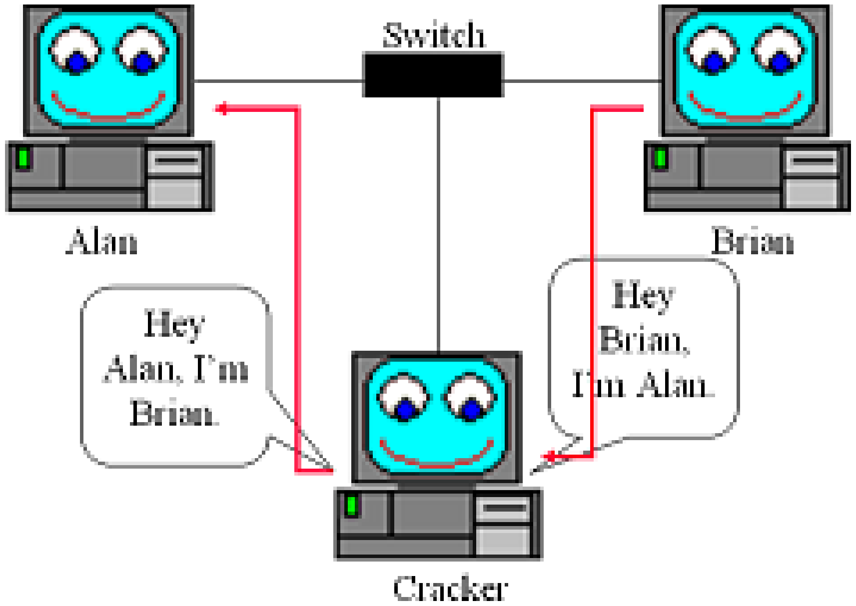
- `rawsend()` uses type SOCK_RAW (see Python help)
  - More on this when we talk about SOCK_STREAM and SOCK_DGRAM
- Did the remote host reply to your request?

TASK 4: ARP Cache Poisoning

- What if the dst of the Ethernet frame was not broadcast?
- What if you didn't use YOUR source IP address within the ARP request?
- What if you used an IP address that's not even within the local network?
- Team up with the person sitting next to you and MITM each other Break their connection to a python web server.
  - Try stuff, break stuff, we can always reboot ;-)

BONUS: Man in the Middle

- Attacker tricks 2 hosts, making them think the attacker is the other host
- Attacker now sees all traffic between the 2 hosts
- ARP Cache Poison attacks only work inside of 1 LAN
- sudo sysctl –w net.ipv4.ip_forward=1



Lesson Review

- Questions
- Summary
- Review

ARP MITM Mitigation

- Static entries (scale of n^2)
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters, ArpCacheLife, ArpCacheMinReferenceLife, ArpUseEtherSNAP, ArpTRSingleRoute, ArpAlwaysSourceRoute, ArpRetryCount
- **arp_accept, arp_ignore, arp_announce, arp_filter, proxy_arp**
- AntiARP (MS)
- ArpStar (Linux 2.6)
- IDS Systems (Snort)

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

Network Layer

Objectives

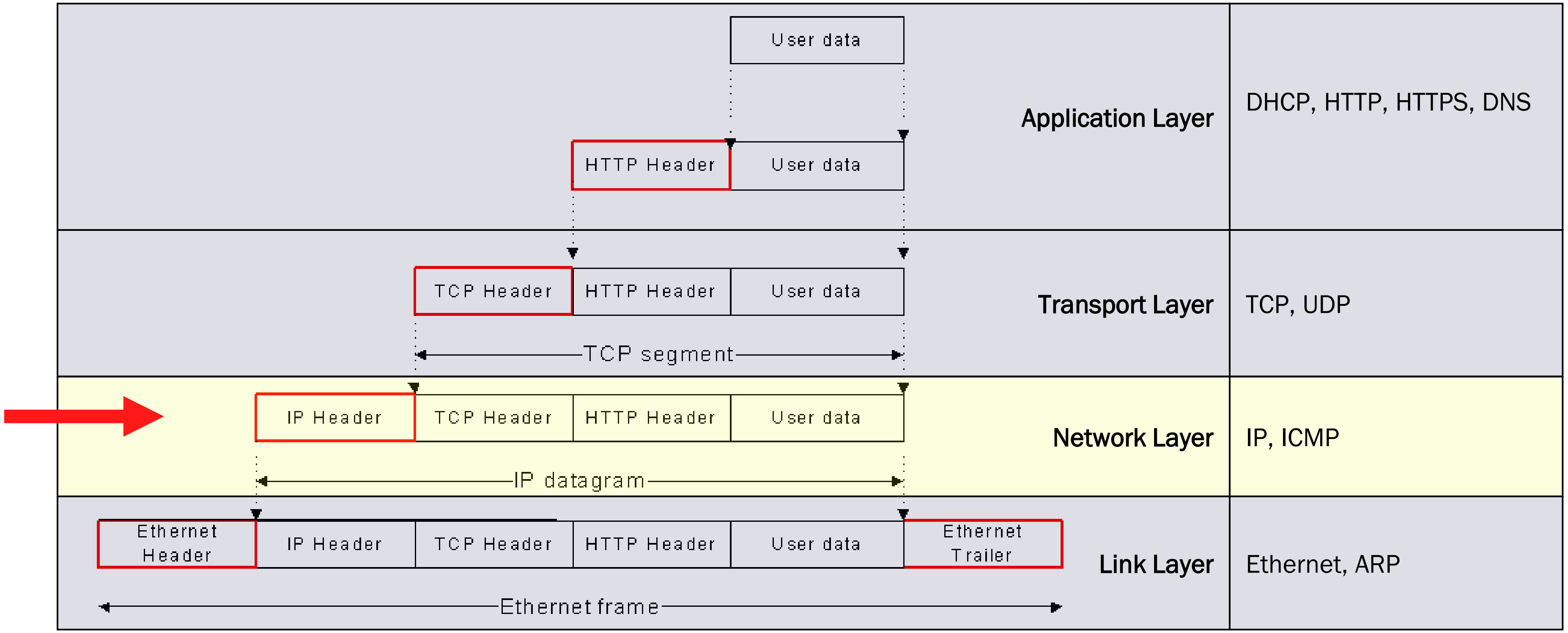
Given a workstation, device, and/or technical documentation, the student will be able to:

- Learning Objective
  - Use common Network Layer protocol specifications to analyze, interpret, and construct network traffic

- Enabling Objectives
  - Describe the differences between IPv4 and IPv6
  - Describe packet fields in terms of size, location, and purpose
  - Describe ICMP message fields in terms of size, location, and purpose
  - Use `ping` and `traceroute` to take a look at their behavior over the network
  - Construct ICMP messages

Network Layer

- Manage the movement of packets around the network
- Make sure packages reach their destination, and report errors if they do not
- Includes IP, ICMP, and IGMP



LINUX
CNO
Programming

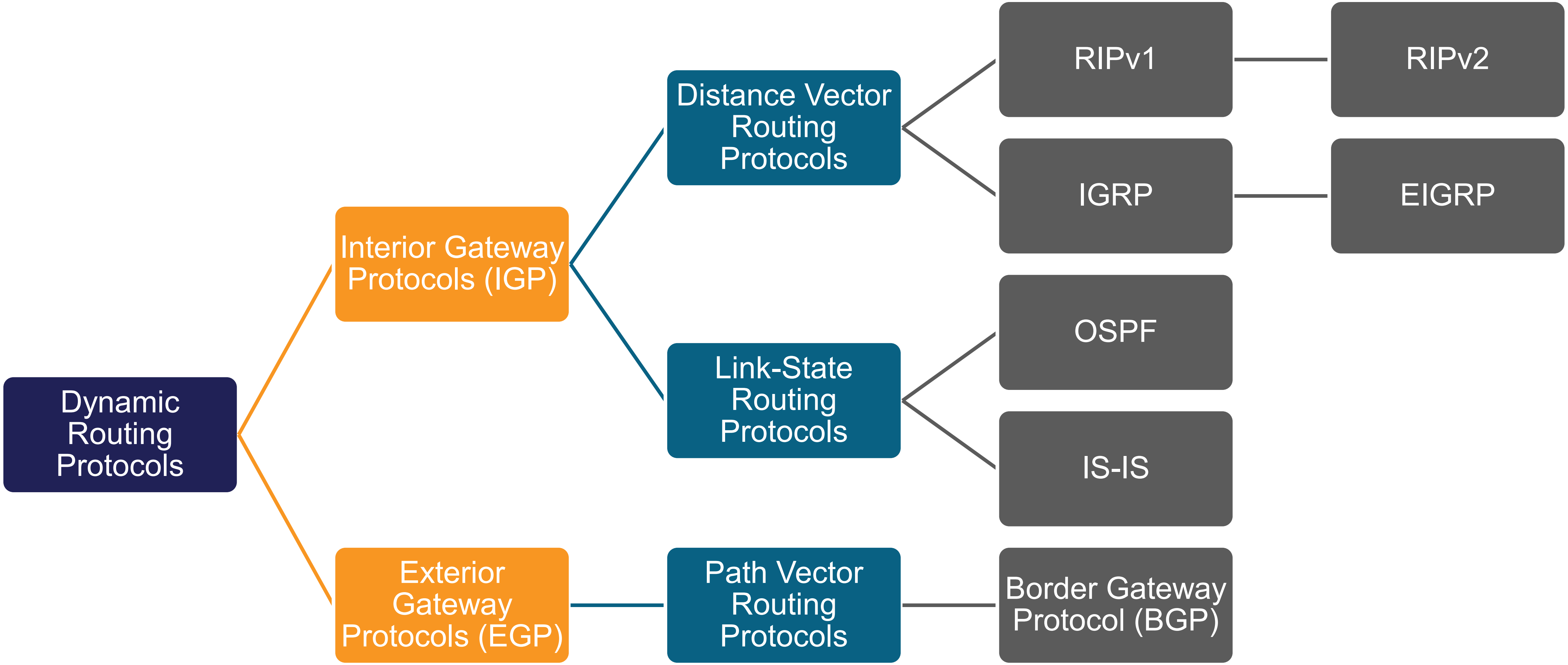
ACTP
ADVANCED CYBER TRAINING PROGRAM

Routing Protocols

Why Routing Protocols

- Network Layer
- Specifies how routers communicate with each other and enables them to select paths between nodes on a network
  - Optimal Routing – Best path
  - Stability of the network
  - Flexibility

Types of Routing Protocols



Dynamic Routing Protocols

Interior Gateway Protocols (IGP)

Exterior Gateway Protocols (EGP)

Distance Vector Routing Protocols

Link-State Routing Protocols

Path Vector Routing Protocols

RIPv1

RIPv2

IGRP

EIGRP

OSPF

IS-IS

Border Gateway Protocol (BGP)

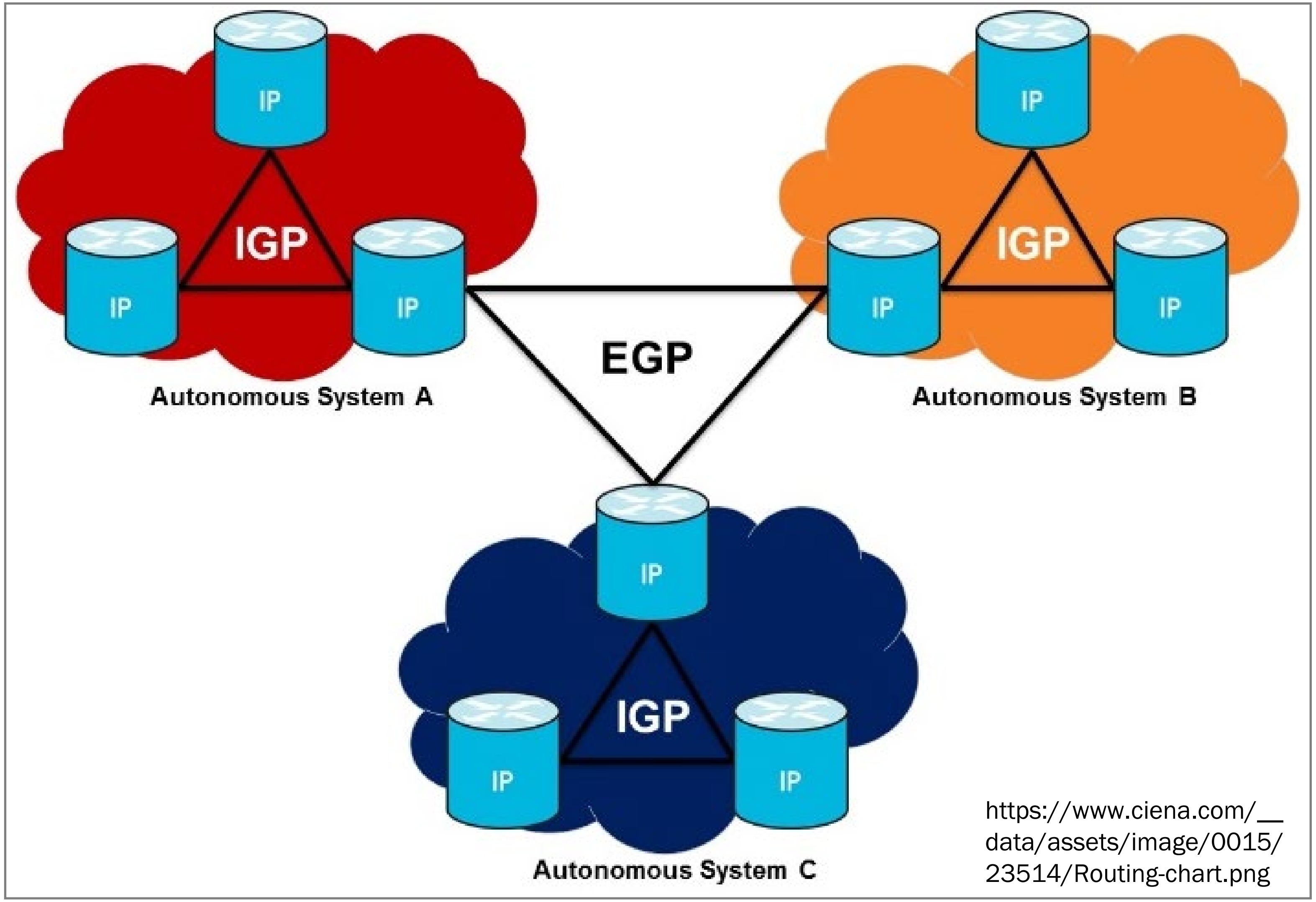
Interior Gateway Protocol (IGP)

- These are protocols used for routing within an Autonomous System (AS).
- Two Types:
  - Distance Vector Routing Protocols - uses hops and distance to decide the best path
    - RIP
    - EIGRP
  - Link-State Routing Protocols - each router has an identical copy of the network which allows each router to calculate its best path
    - OSPF
    - IS-IS

Exterior Gateway Protocols (EGP)

- Routing between autonomous systems (AS).
  - Border Gateway Protocol (BGP) is the Internet's official routing protocol and the only EGP that is currently operational.

Routing Chart



https://www.ciena.com/__data/assets/image/0015/23514/Routing-chart.png

Open Shortest Path First (OSPF)

- Find best path between the source and the destination router using its own shortest path first (SPF) algorithm.
- Protocol number 89.
- Uses multicast address 224.0.0.5 for normal communication and 224.0.0.6 for updating the designated router (DR)/Backup Designated Router (BDR).

RFC:
2328

Open Shortest Path First (OSPF)

RFC:
2328

Messages

- Hello message
- Database Description (DBD)
- Link state request (LSR)
- Link state update (LSU)
- Link state Acknowledgment
- Link state advertisement (LSA)

States

- Down
- Init
- 2Way
- Exstart
- Exchange
- Loading
- Full

Enhanced Interior Gateway Routing Protocol (EIGRP)

- Uses metrics to find out the best path between two layer 3 devices
- Protocol number 88
- It is a Cisco-proprietary protocol. Only Cisco routers will be able to interact. Non-Cisco routers will be unable to use or understand EIGRP.
- EIGRP uses five different packets, which are as follows:
  - Hello
  - Update
  - Query
  - Reply
  - ACK (Acknowledgement)

RFC:
7868

Routing Information Protocol (RIP)

- Uses hop count to find the best path between the source and destination network.
- Uses port number 520
- RIP was first defined in RFC 1058 as a first-generation routing protocol for IPv4.
- RIPv1 possesses the following qualities:
  - Number of hops is the path selection metric
  - Every 30 seconds, routing updates are transmitted (255.255.255.255)
  - Greater than 15 hops is considered too far

RFC:
2453

Routing Information Protocol (RIP)

- In 1993, RIPv1 evolved into RIPv2 with added enhancements
  - Security: includes authentication for securing routing table updates
  - Supports CIDR because routing updates include the subnet mask
  - Improved efficiency: updates forwarded to the multicast address 224.0.0.9 instead of the broadcast address 255.255.255.255
  - Manual route on any interface is supported
  - Updates are contained in a UDP segment - port 520 (src and dst)
- The IPv6-enabled version of RIP was introduced in 1997.
- RIPng is an extension of RIPv2 restricted to 15 hops
  - This hop count limitation renders RIP unsuitable for larger networks.

RFC:
2453

Intermediate System-to-Intermediate System (IS-IS)

- Runs the Dijkstra SPF algorithm to find the best path to each destination, which is installed in the routing table.
- IS-IS was originally developed for Connectionless-mode Network Service (CLNS), not IP. Later, it was adapted so that it could also route IP
- IS-IS is directly on top of an Ethernet header. It’s not encapsulated in an IP packet like other routing protocols

Ethernet Header	IS-IS Header	IS-IS Data	FCS

- Often used on large service provider networks

RFC:
5305

Border Gateway Protocol (BGP)

- Used to exchange routing information for the internet and is the protocol used between different Autonomous Systems (AS)
- Can connect together any internetwork of AS
  - Only requirement: each AS has at least one router able to run BGP
- Constructs an autonomous systems’ graph based on the information exchanged between BGP routers.
- BGP supports these session types between neighbors:
  - Internal (iBGP) - Runs between routers in the same autonomous system
  - External (eBGP) - Runs between routers in different autonomous systems
- BGP was chosen over OSPF since it allows device designers and owners greater flexibility and control

RFC:
4271

LINUX
CNO
Programming

ADVANCED CYBER TRAINING PROGRAM

Internet Protocol Overview

IPv4 Overview

- IPv4 address space: 32-bit
    - 4 octets, 8 bits each
    - 4+ billion
- Address Resolution using the Address Resolution Protocol
- Network Address Translation
- Classful addressing space (A..E)
    - Private Reserved Addressing (10.*.*.*, 192.168.*.*, etc.)
- Variable length subneting
- Ethernet Broadcast: ff:ff:ff:ff:ff:ff

RFCs:
791, 826, 917

Classful Subnetting

Class	Start	End	Default Subnet Mask	CIDR
Class A	0.0.0.0	127.255.255.255	255.0.0.0	/8
Class B	128.0.0.0	191.255.255.255	255.255.0.0	/16
Class C	192.0.0.0	223.255.255.255	255.255.255.0	/24
Class D (multicast)	224.0.0.0	239.255.255.255		
Class E (reserved)	240.0.0.0	255.255.255.255		

Subnetting

192 . 168 . 64 . 64 /24

11000000 . 10101000 .01000000 . 01000000 /24

255 . 255 . 255 . 0

11111111 . 11111111 . 1111111 . 00000000

Net ID: 192.168.64.0
Broadcast Addr: 192.168.64.255
Subnet: 255.255.255.0
# of addrs: 256

Subnetting

192 . 168 . 64 . 64 /26
11000000 . 10101000 . 01000000 . 01000000 /26
255 . 255 . 255 . 192
11111111 . 11111111 . 11111111 . 11000000

Net ID: 192.168.64.64
Broadcast Addr: 192.168.64.127
Subnet: 255.255.255.192
# of addrs: 64

Subnetting

10 . 0 . 0 . 0 /12

00001010 . 00000000 . 00000000 . 00000000 /12

255 . 240 . 0 . 0

11111111 . 11110000 . 00000000 . 00000000

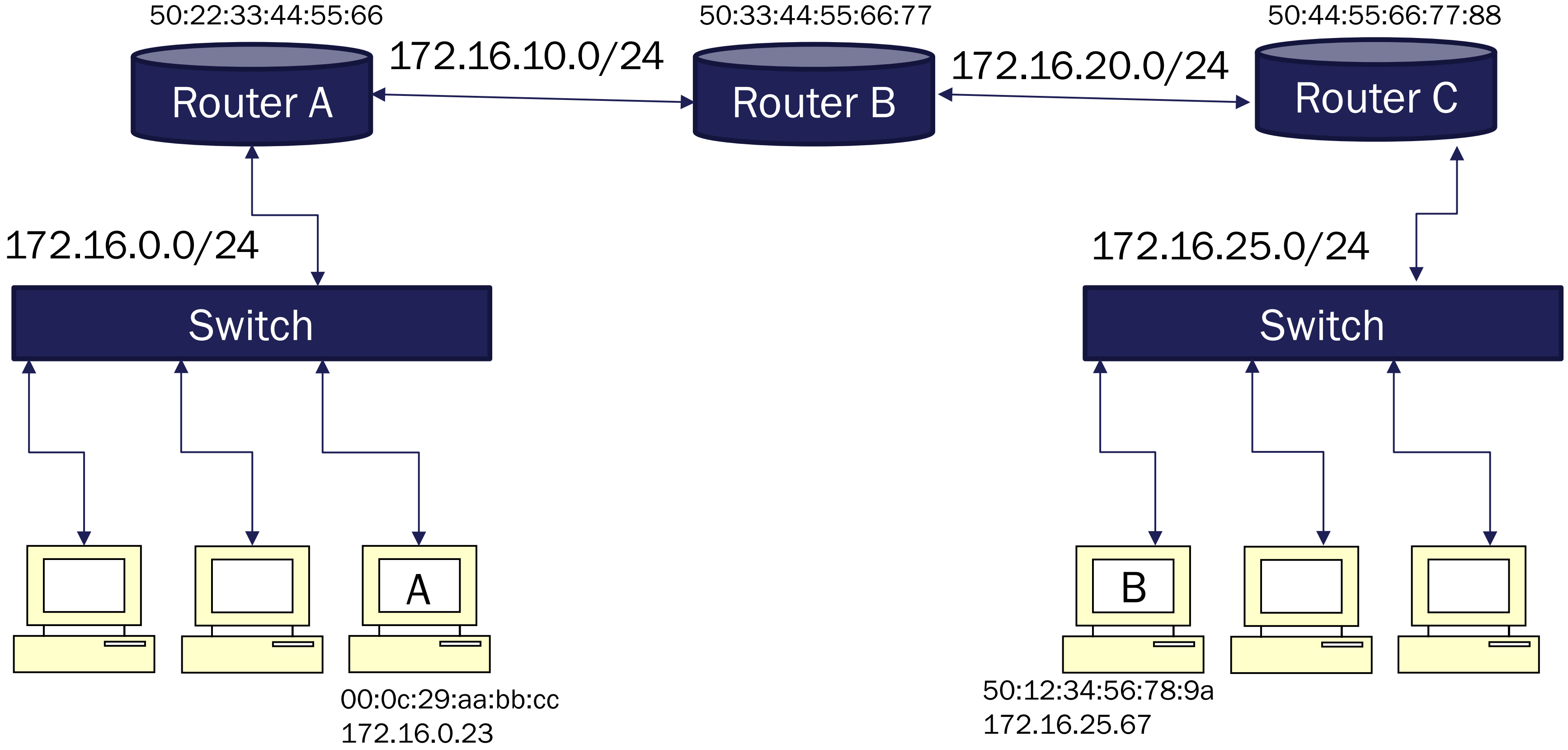
Net ID: 10.0.0.0
Broadcast Addr: 10.15.255.255
Subnet: 255.240.0.0
# of addrs: 1048576

Subnetting

- Now you try a few:
  - Find the subnet mask
  - Find the broadcast address
  - Find the number of ip address in that subnet
  - Total number of ip address in the subnet
    - 172.20.10.67/27
    - 172.72.6.0/25
- Bonus: Write a program that can take a CIDR notation and print out the 4 bullet points of information above.

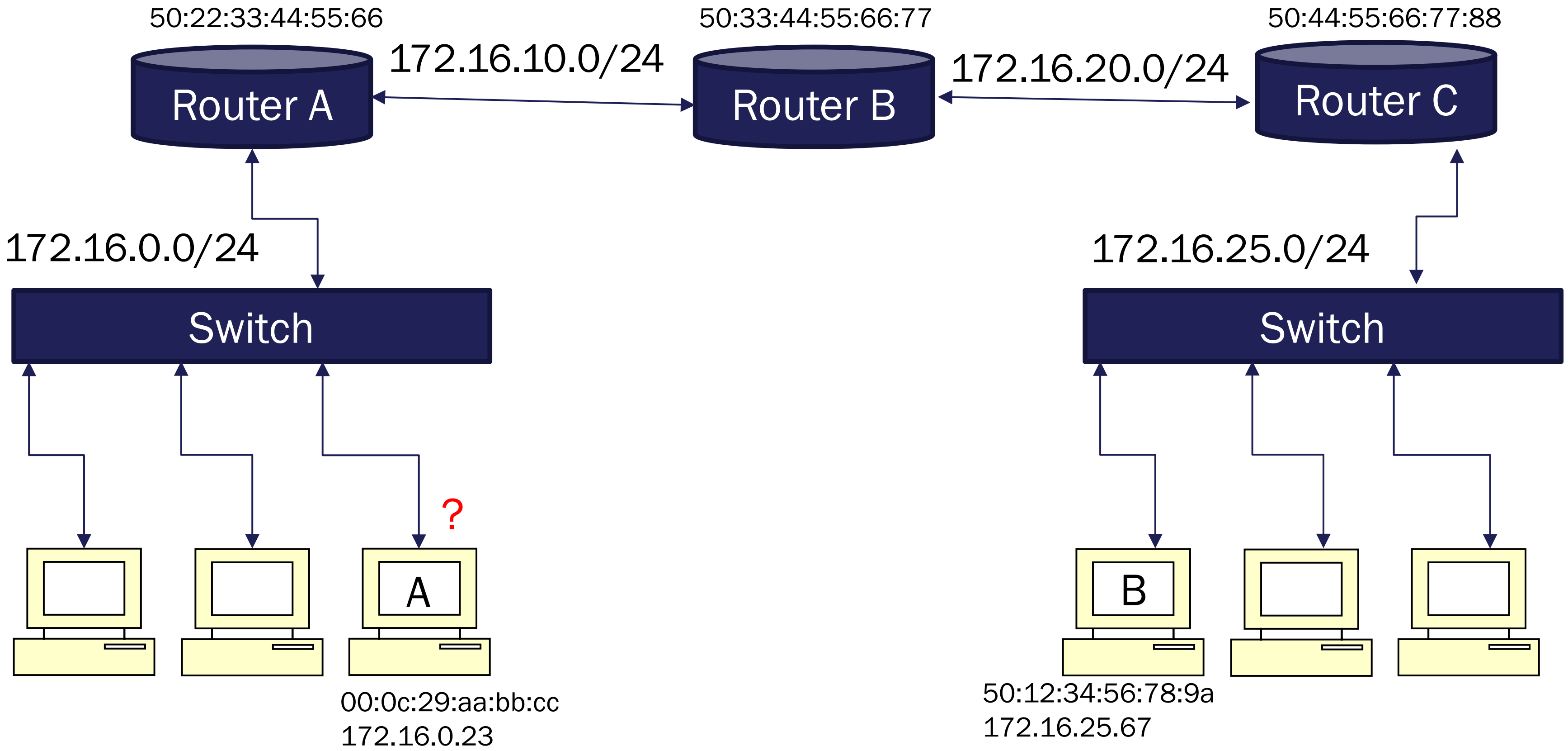
IP Routing

- In the network layer we are now able to move across subnets. Let’s look at how that happens



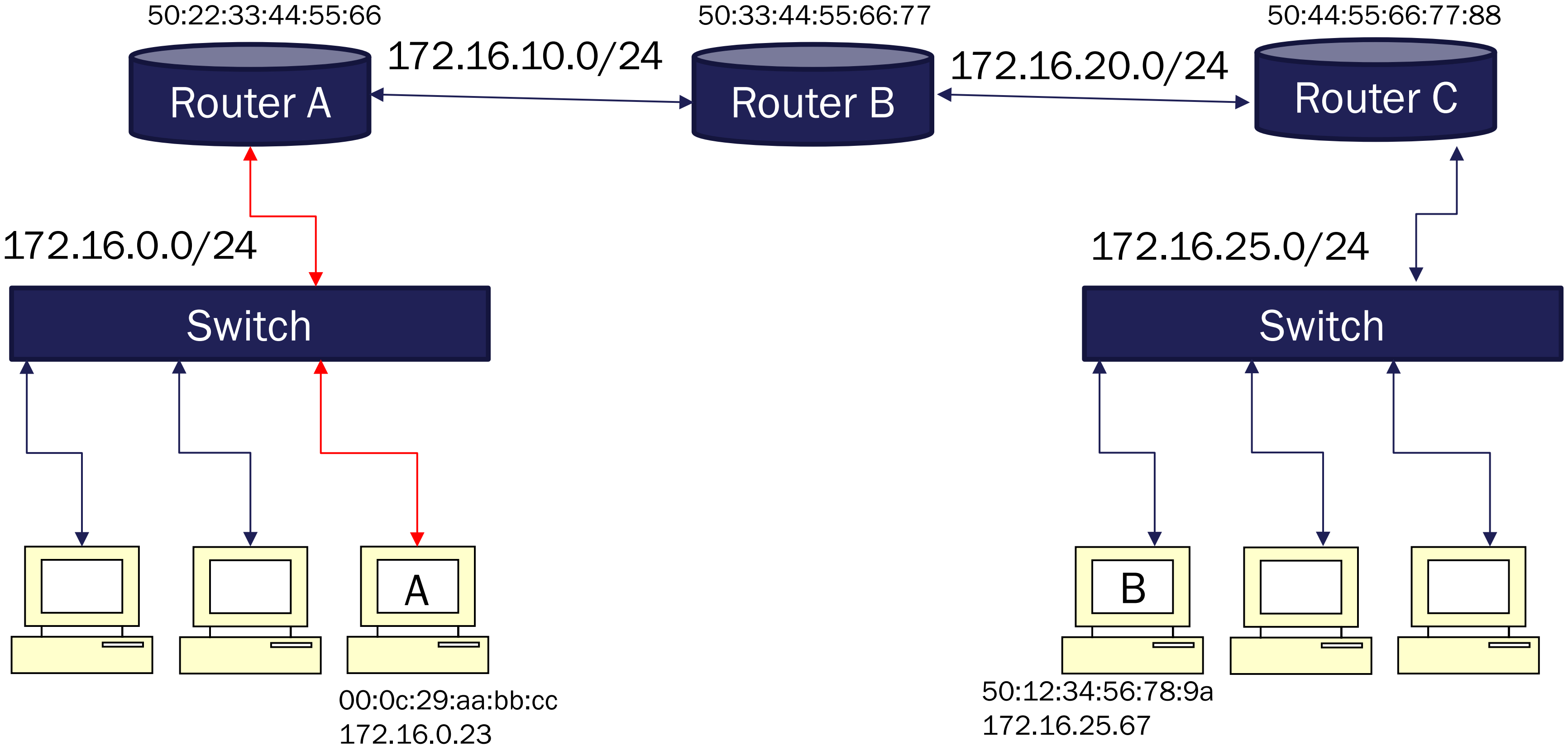
IP Routing

- Host A asks, Is the IP address I am trying to send to in my subnet?
  - If so check arp cache or send arp request
  - If not then send to default gateway



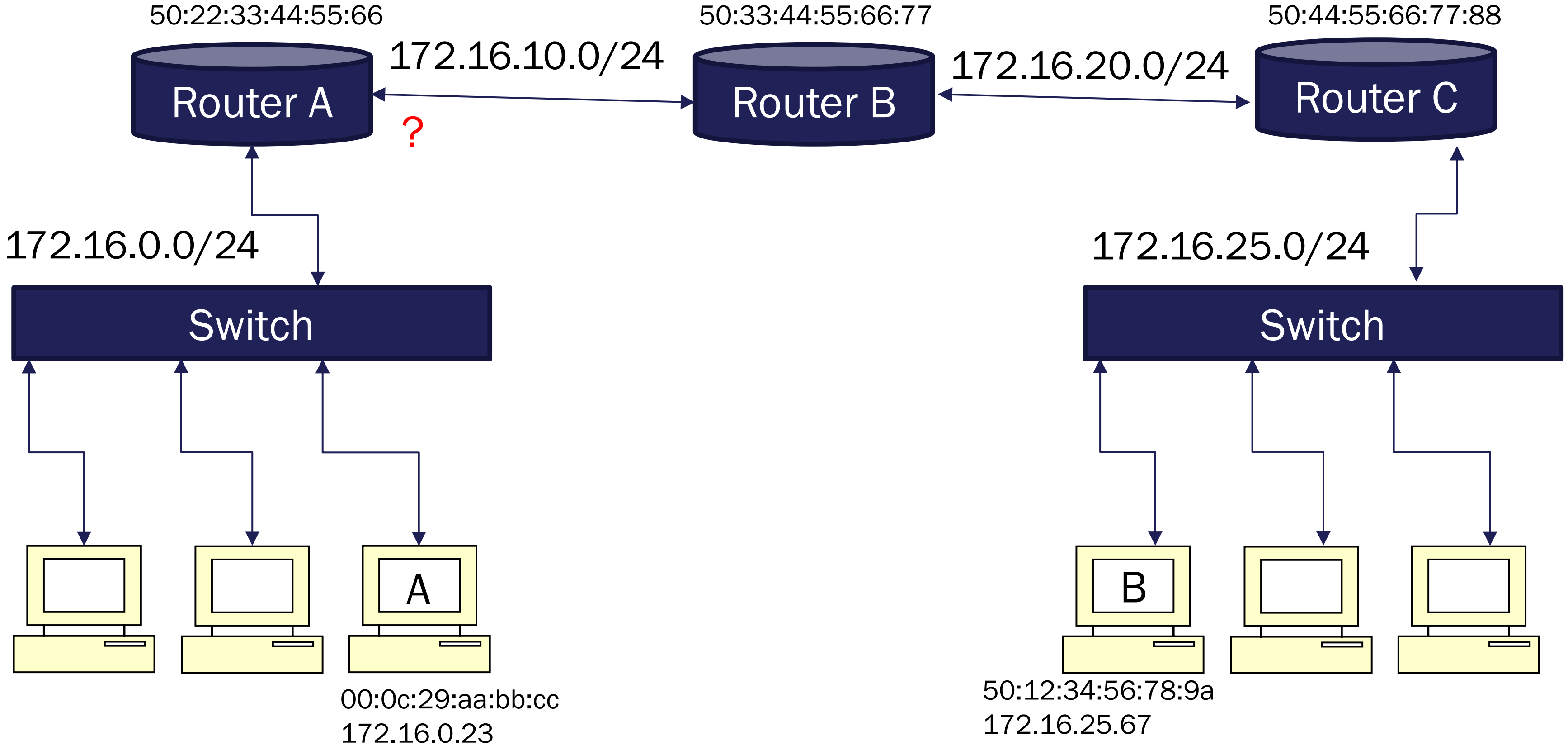
IP Routing

- Host B is not in the subnet, so, send a packet to the default gateway. We check the arp cache or send arp request for mac address of the default gateway and send the packet there



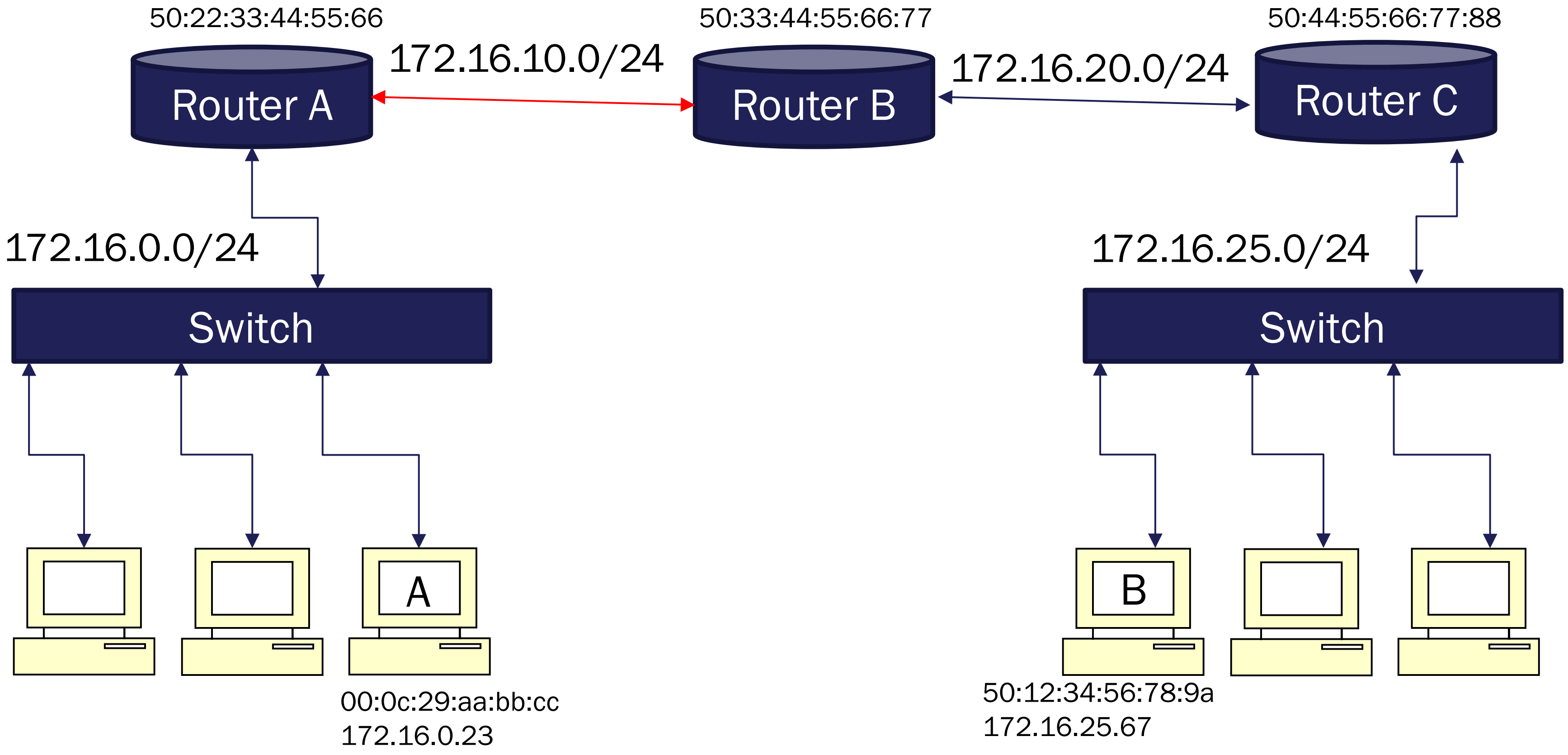
IP Routing

- Router A receives the packet and checks it’s routing table. If it does not have a route for the IP address it will send it to it’s default gateway



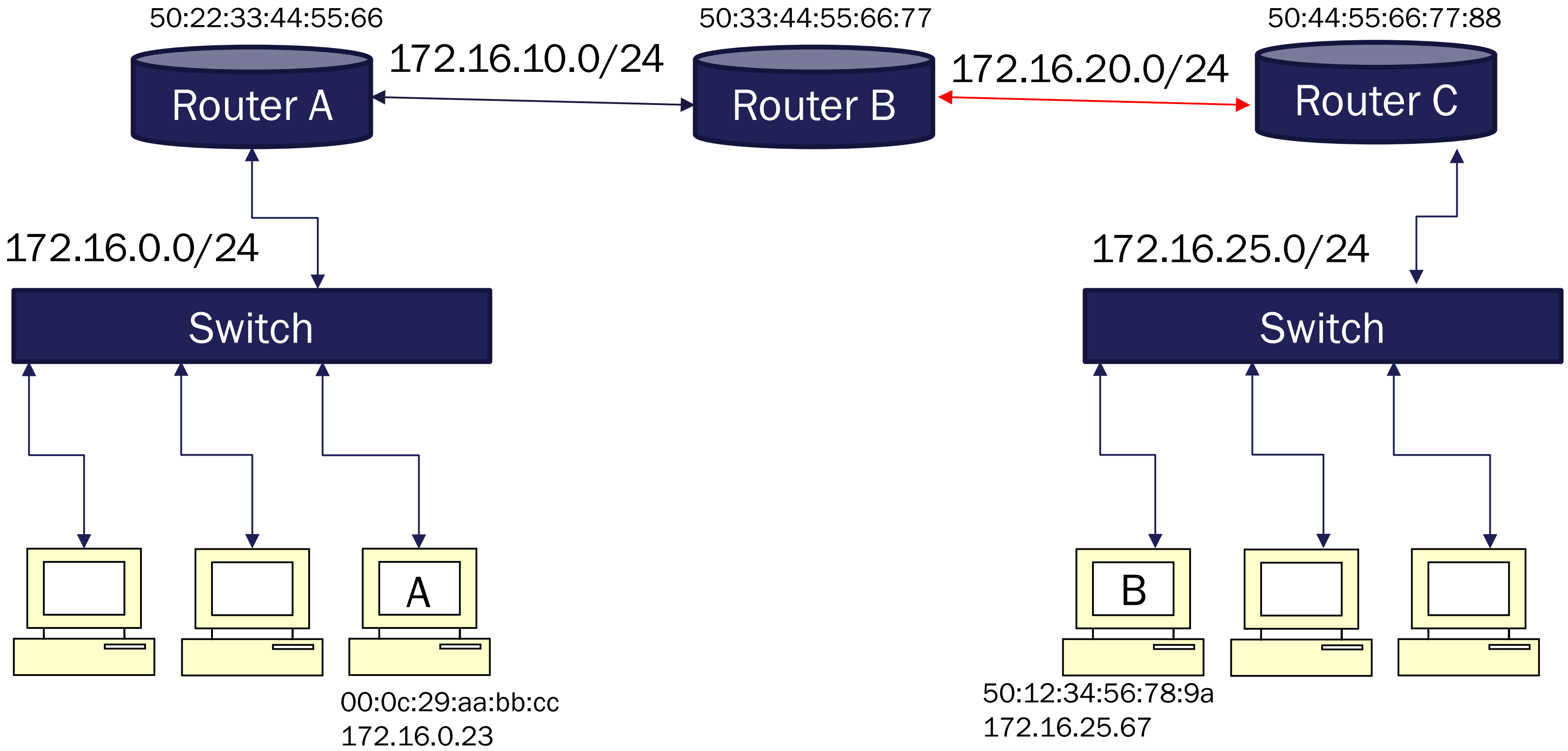
IP Routing

- If a route is found it will send the packet on that interface to the next hop



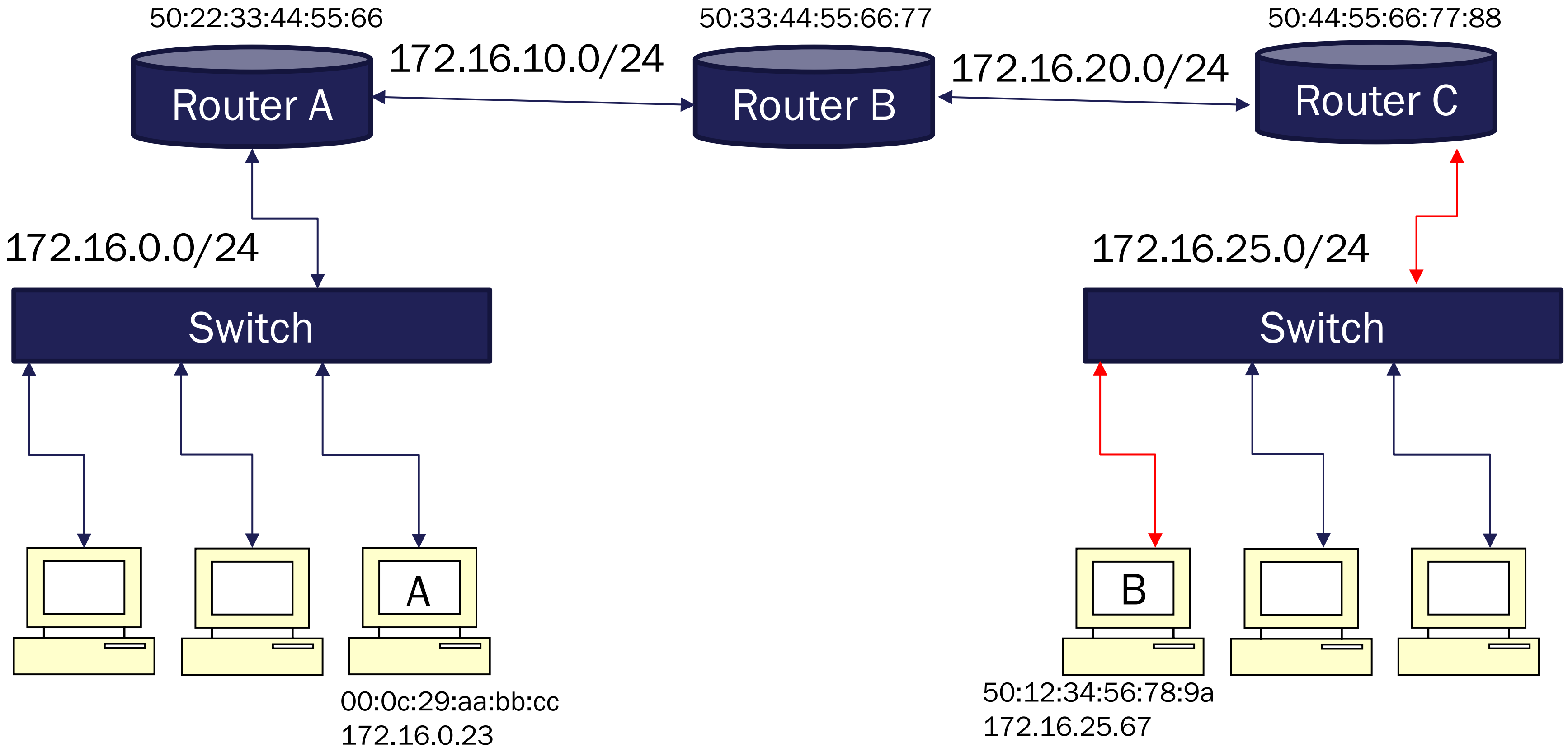
IP Routing

- Router B repeats the same process



IP Routing

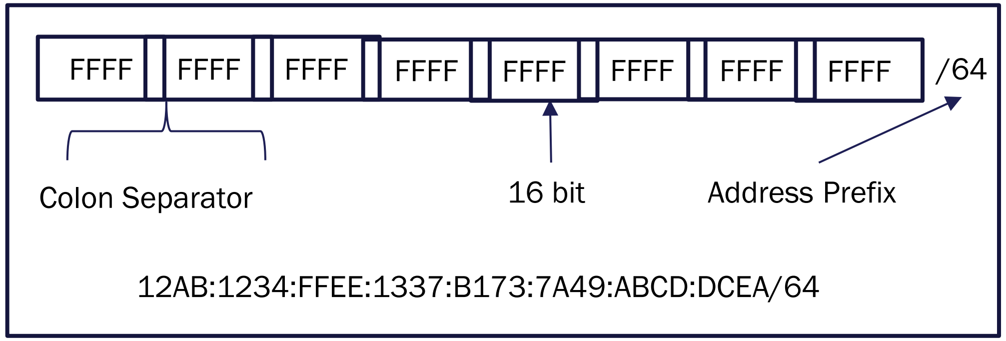
- Finally, router C sees that it has a route to the destination IP address of the packet. It will check it’s ARP cache or send an ARP request and then forward the packet to host b



IPv6 Overview

- 128 bit addressing
  - IPv4: 4,294,967,296
  - IPv6: 340,282,366,920,938,463,463,374,607,431,768,211,456
- Preferred Notation: 8 x 16 bit blocks
- Unicast: Interface
  - Loopback Address is ::1
  - Any addr is ::
- Multicast: Set of interfaces
- Anycast: Set of interfaces
- Local-link address
- Address representation
  - Double-colon shortcut

Compare to IPv4:
128 bit vs. 32 bit
Multiple IPs per interface
RFCs:
8200, 4291



12AB:1234:FFEE:1337:B173:7A49:ABCD:DCEA/64

Shortening IPv6 addresses

- A single ipv6 address can be represented many ways
  - 2001:0db8:0000:0000:0001:0000:0002:0001
  - 2001:db8:0:0:1:0:2:1
  - 2001:0db8:0:0:1:0:2:1
  - 2001:db8::1:0:2:1
  - 2001:db8::0:1:0:2:1
  - 2001:0db8::1:0:2:1
  - 2001:db8:0000:0:1:0:2:1

RFCs:
4291, 5952

Shortening IPv6 addresses

- So rfc 5952 specifies some general rules for shortening
- The “proper” way to write the address would be:
    - 2001:db8::1:0:2:1

IPv6 Address Type Representation

- Format Prefixes 001 through 111 (except Multicast) are required to have a 64 bit interface (EUI-64 format)
- Loopback addresses use the 0000 0000 space

Compare to IPv4:
Private vs. Public IPs
Broadcast Addresses
RFCs:
8200, 4291

ALLOCATION	PREFIX (BINARY)	PREFIX (HEX)	FRACTION OF ADDRESS SPACE
Reserved	0000 0000	00	1/256
Aggregatable Global Unicast Addresses	001	{2,3}	1/8
Link-Local Unicast Addresses	1111 1110 10	FE{8-B}	1/1024
Multicast Addresses	1111 1111	FF	1/256

Transitioning to IPv6 Addresses

- IPv4-mapped IPv6 addresses:

RFCs:
4291, 5156

80 bits	16 bit	32 bit
0	FFFF	IPv4 Address

Transitioning to IPv6 Addresses

- Teredo tunneling is a transition technology, developed by Microsoft, which allows IPv6 enabled host to use full IPv6 capabilities even if they are on an IPv4 network
- It works by encapsulating and IPv6 packet in and IPv4 UDP packet.
- Teredo relays are then used to un-encapsulate the packet and forward it on based on the contents of the IPv6 packet

32 bits	32 bits	16 bit	16 bit	32 bit
Prefix	Server IPv4	Flags	Port	Client IPv4 Address

Global Unicast Address Format

Global routing prefix	Subnet ID	Interface ID

Compare to IPv4:
IPv4: (ie. 209.93.100.2)

RFCs:
4291, 3587

- Global routing prefix (n bits): Typically a hierarchically-structured value assigned to a site
- Subnet ID (m bits): An identifier of a subnet within the site
- Interface id (Usually 64 bits): An id specific to the node

Local-Use IPv6 Unicast Addresses

- Used for addressing on a single link
- Routers must not forward any packets with link-local source or destination addresses

Compare to IPv4:
Private IP addresses

RFCs:
4291, 3587

FP	0	Interface ID

- FP (10 bit): Format Prefix (1111111010) (FE8)
- Zeros (54 bit)
- Interface (64 bit): EUI-64 format

IPv6 Anycast Addresses

- Assigned to more than one interface/node
- Sends packets to the “nearest” interface with assigned address
- Assigned from unicast address space (in unicast address format)

Subnet prefix (n bits)	128-n bits

- Restrictions:
  - Must not be assigned to an IPv6 host (only routers)

Compare to IPv4:
Broadcast addresses were eliminated
RFCs:
4291, 3587, 1546

IPv6 Multicast Addresses

- Assigned to more than one interface/node
- Sends packets to a group of nodes

FP	F	S	Group ID

Compare to IPv4:
Broadcast addresses were eliminated
RFCs:
4291, 3587

- FP (8 bit): 1111 1111 (signifies start of an address)
- F (4 bit): 000T (first three reserved)
- T=0: indicates permanent multicast address
- T=1: indicates non-permanent multicast address
- S (4 bit): limits scope

1	Node-local scope
2	Link-local scope
5	Site-local scope
8	Organization-local scope
E	Global-scope

IPv6 Multicast Addresses *(continued)*

Compare to IPv4:
Broadcast addresses were eliminated
RFCs:
4291, 3587

- Non-permanently assigned addresses
  - Meaningful only in given scope
- Permanently-assigned multicast addresses (example)
  - FF01:0:0:0:0:0:0:101: NTP servers on same node
  - FF02:0:0:0:0:0:0:101: NTP servers on same link
  - FF05:0:0:0:0:0:0:101: NTP servers on the same site
  - FF0E:0:0:0:0:0:0:101: NTP servers on the internet
- Restrictions
  - Multicast addresses must not be used as source addresses
  - Ethernet Multicast: 33:33:00:00:00:01

IPv6 Multicast Addresses *(continued)*

TYPE	MULTICAST ADDRESS
All Nodes Addresses	FF01:0:0:0:0:0:0:1 FF02:0:0:0:0:0:0:1
All Routers Addresses	FF01:0:0:0:0:0:0:2 FF02:0:0:0:0:0:0:2 FF05:0:0:0:0:0:0:2
Solicited-Node Address	FF02:0:0:0:0:1:FFXX:XXXX

Compare to IPv4:
Broadcast addresses were eliminated
RFCs:
4291, 3587

- Solicited-Node addresses are computed by taking the last 24 bits and appending to the prefix
- A node is required to join the associated Solicited-Node address for every unicast and anycast address assigned

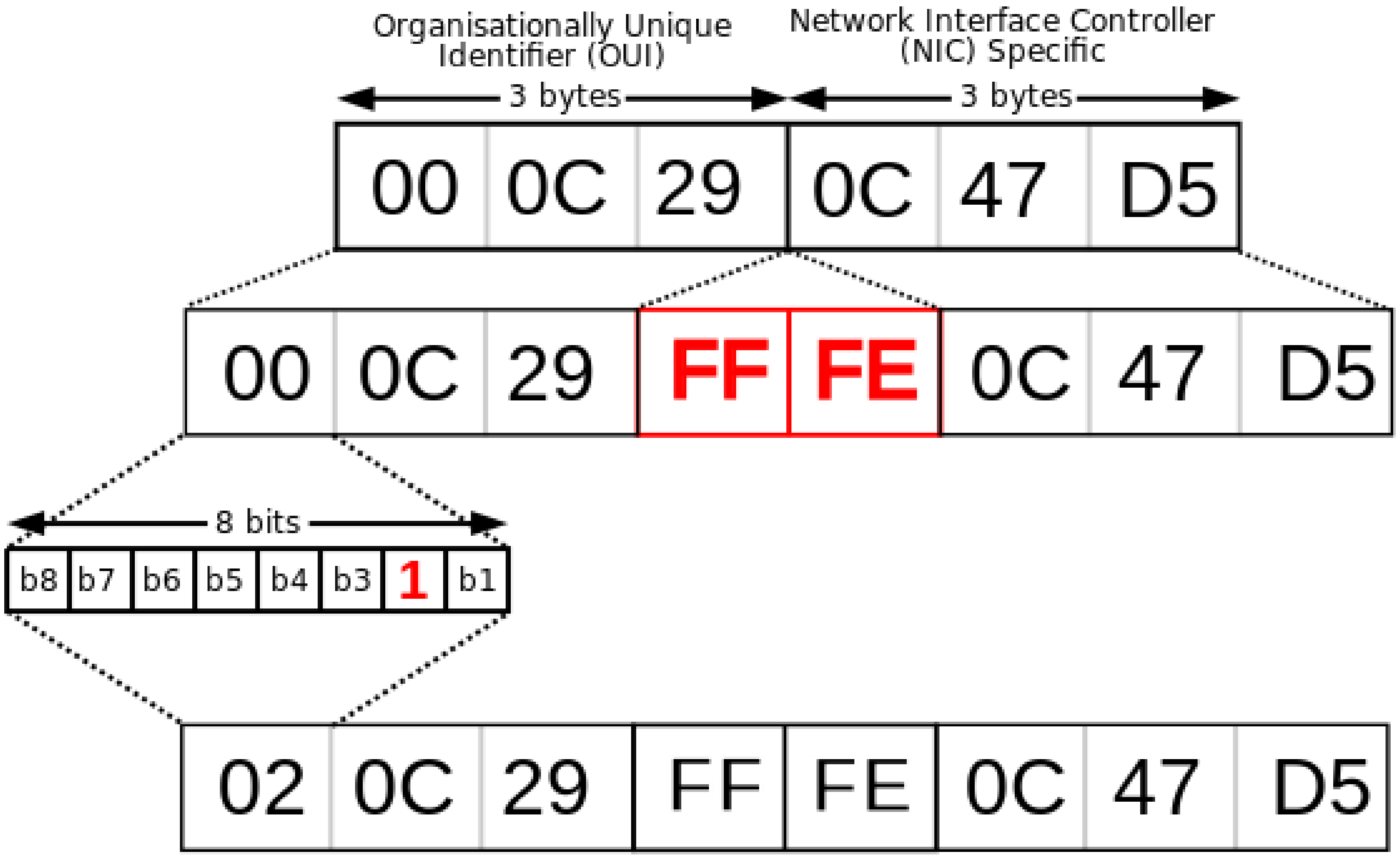
Extended Unique Identifier (EUI-64)

- 64-bit unique identifier
- Company ids assigned by IEEE
  - 24-bit / 36-bit company id value
  - 40-bit / 30-bit assignment value (by company)
- Can encapsulate smaller EUI-48 and MAC-48 addresses to create EUI-64 identifier (interface identifier)
  - Uses FFFF or FFFE to separate MAC-48 OUI and extension

Compare to IPv4:
MAC Addresses

Source:
IEEE EUI-64 Guidelines

MAC-48 to EUI-64 Conversion



Compare to IPv4: MAC Addresses

Source:

By AKRAM.ABOU - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=30091884

Stateless Address Autoconfiguration (SLAAC)

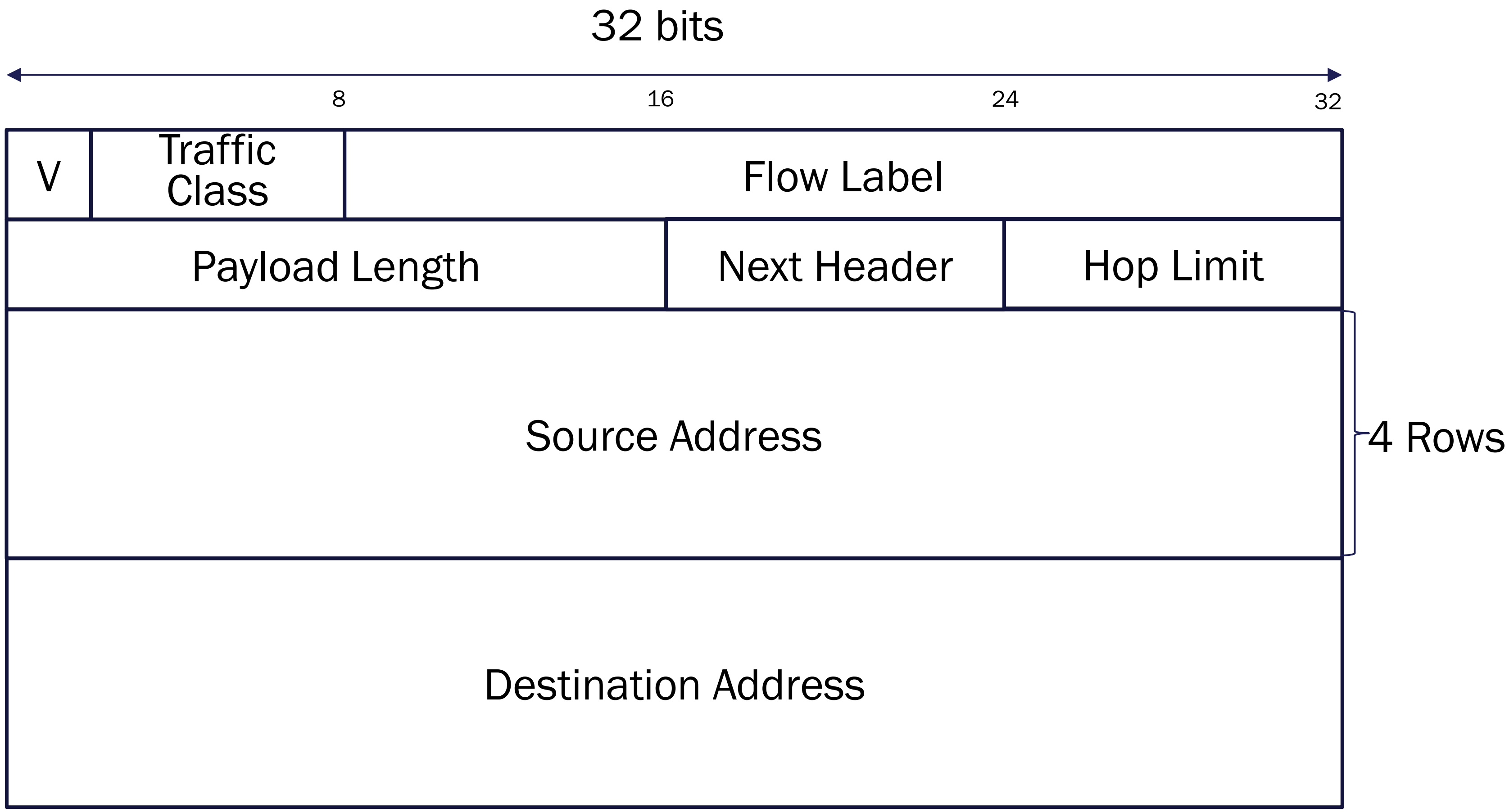
- Specifies steps hosts take to autoconfigure its interfaces for IPv6
- Host can also use stateful address configuration through protocols like DHCPv6
- Uses a duplicate address detection algorithm for determining if that address already exists
- Host can also use stateful address configuration through protocols like DHCPv6
- Both SLAAC and DHCPv6 can be used simultaneously

Source:
RFCs 1971,, 4862, 7527, 8415

LINUX
CNO
Programming

IP Header Fields

IPv6 Data Fields



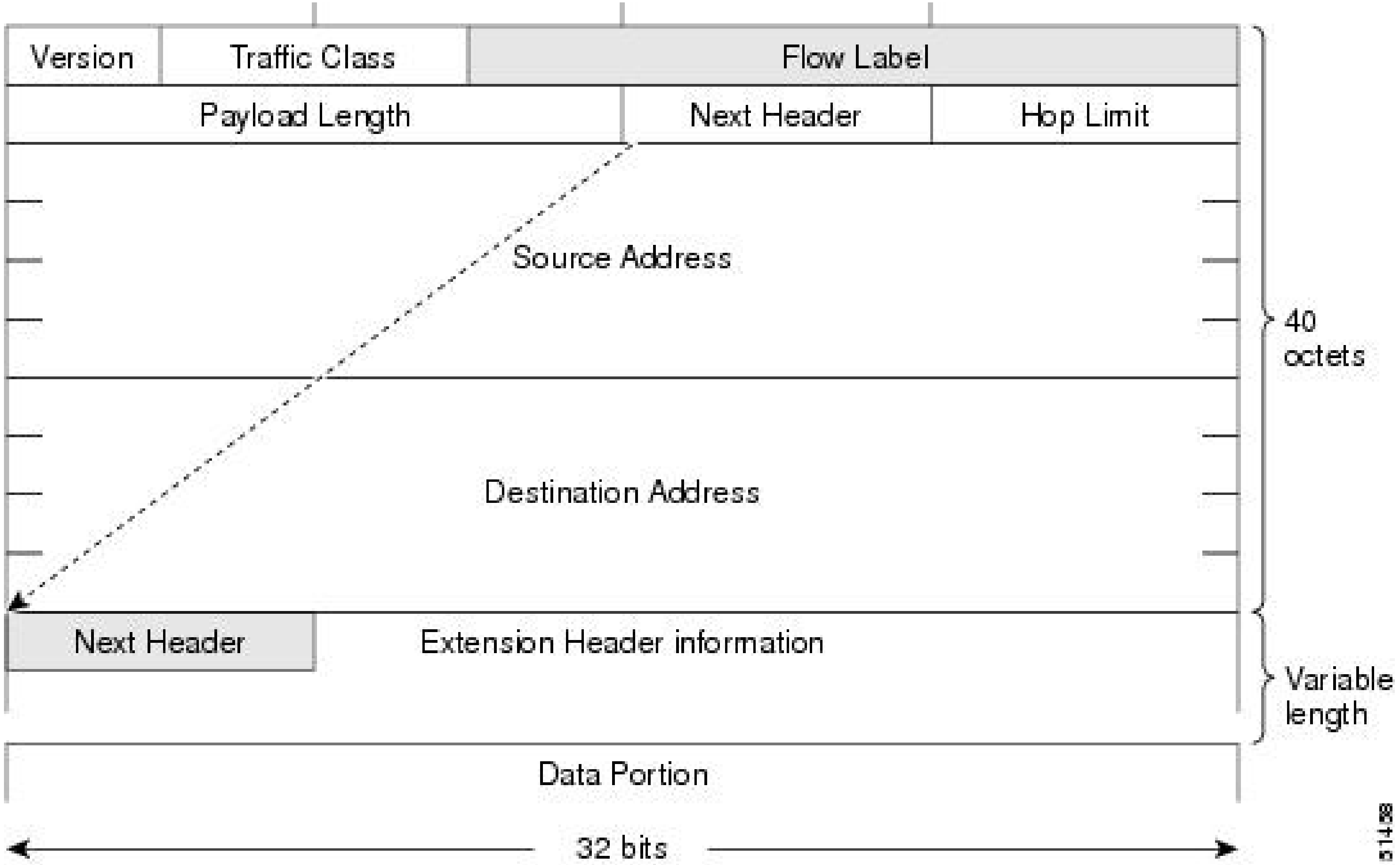
Version and Header Length

- (V)ersion (4 bits): 6
- Traffic Class (8 bits): Set priorities (QOS)
- Flow Label (20 bits): Request Special Handling by routers
- Payload Length (16 bit unsigned int): length of the IPv6 payload, i.e. the rest of the packet following this IPv6 header, in octets (aka bytes)
- Next Header (8 bits): Identifies the type of header immediately following the IPv6 header – uses the same values as the IPv4 protocol field (RFC 1700)
- Hop Limit (8 bits unsigned int): Decremented by 1 by each node that forwards the packet. Packet is discarded
- Can carry zero, one or more extension headers

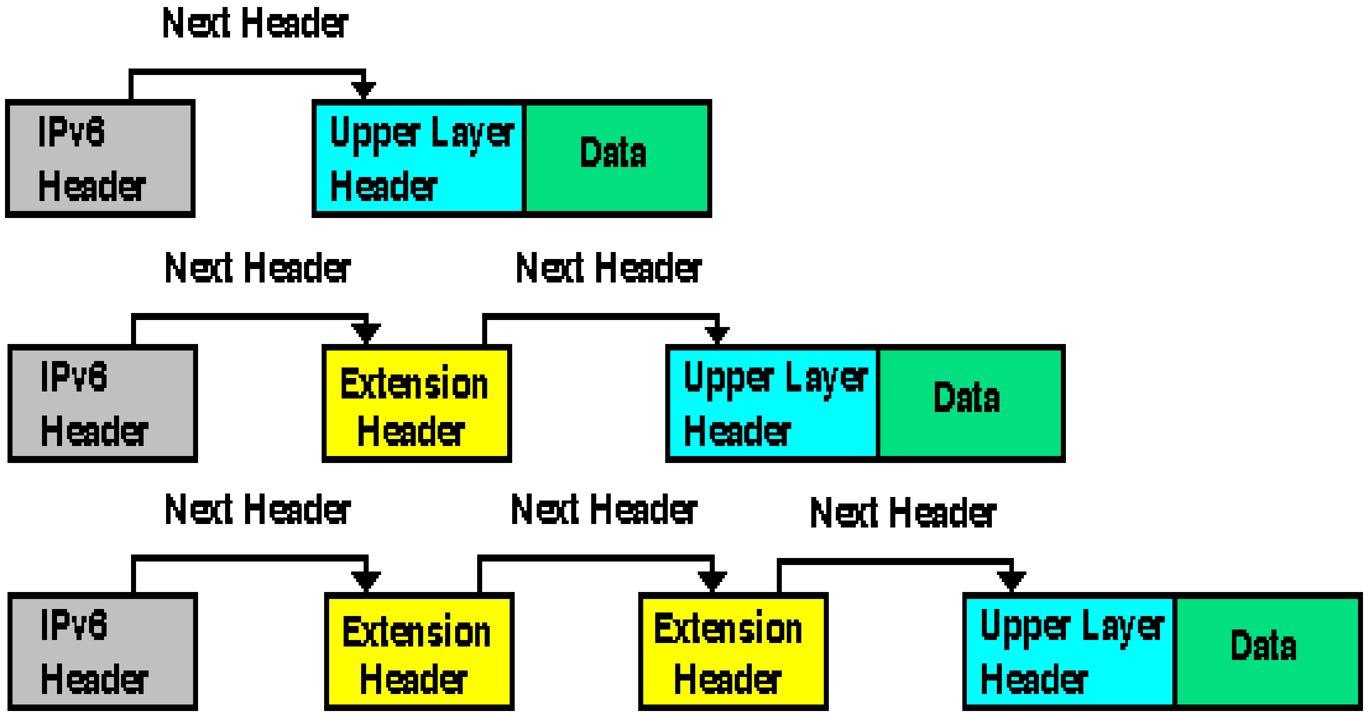
Compare to IPv4:
IPv4 header in Appendix

RFCs:
8200, 1700

Frames


Version
Traffic Class
Flow Label
Payload Length
Next Header
Hop Limit
Source Address
Destination Address
Next Header
Extension Header information
Data Portion
40
octets
Variable
length
32 bits
51438

Extension Headers



Compare to IPv4:
None (New)

RFCs:
8200

Extension Headers *(continued)*

- Have to examine and process in sequence
- Extension Headers / Order / Sandwich:
  - IPv6 Header
  - + Hop-by-hop options (always first, if present)
  - + Destination Options
  - + Routing
  - + Fragment
  - + Authentication
  - + Encapsulating Security Payload
  - + Destination Options Header
  - Upper-Layer Header

Compare to IPv4:
None (New)

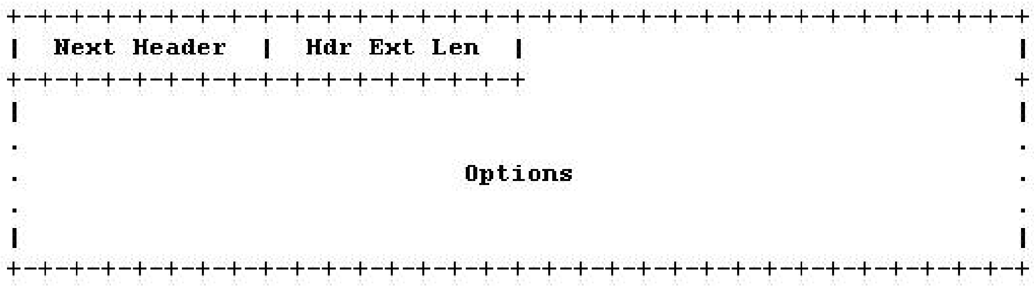
RFCs:
8200

Type-length-value Encoding

- Extensions header format
- Variable Number of Type-length-value (TLV) encoded options
- TLV in multiple of 8 octets (or bytes)
- Sequence of options must be processed in order

Compare to IPv4:
None (New)

RFCs:
8200



```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |  Hdr Ext Len  |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
.                                                               .
.                            Options                            .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IPv6 Extension Header

Header Values

VALUE	PROTOCOL / EXT HDR
00	Hop-by-Hop
01	ICMPv4
02	IGMPv4
04	IP in IP encapsulation
06	TCP
08	EGP
17	UDP
41	IPv6
43	Routing Extension Header
44	Fragmentation Extension Header
46	Resource Reservation Protocol
50	Encrypted Security Payload (ESP) Extension Header

Compare to IPv4:
None (New)

RFCs:
8200

Header Values

VALUE	PROTOCOL / EXT HDR
51	Authentication Header (AH) Extension Header
58	ICMPv6
59	No Next Header
60	Destination Options Extension Header

Compare to IPv4:
None (New)

RFCs:
8200

Jumbograms

- Send packages up to 4GB (32-bit)
- Relevant only with MTU of 65,575+ octets
- Carried in IPv6 Hop-by-Hop options
- Replaces the Payload Length field
  - Affects checksum calculation
- Interacts with TCP
- Problem for UDP… but possible

Compare to IPv4:
None (New)

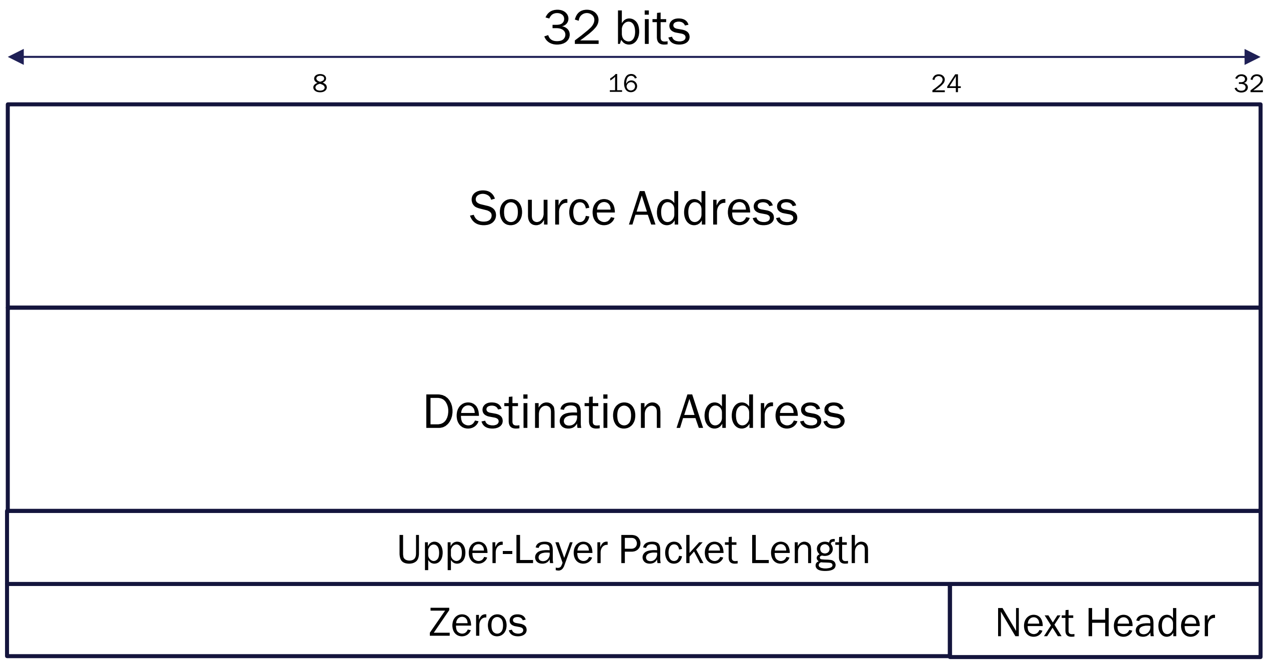
RFCs:
2675

Upper Layer Checksums

- Verifies the integrity of the packet (bit level)
- IPv6 no longer requires checksums at IP level
- Pseudo IP Header used by upper layer protocols:

Compare to IPv4:
No more checksum at IP level

RFCs:
1071, 1624, 8200



8	16	24	32
Source Address			
Destination Address			
Upper-Layer Packet Length			
Zeros			Next Header

LINUX
CNO
Programming

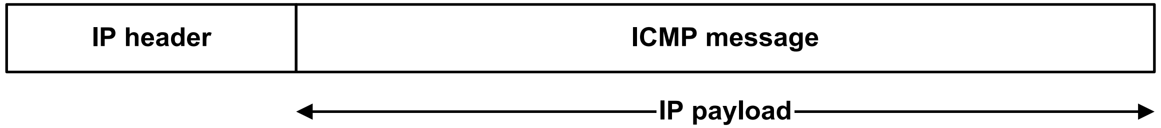
ACTP
ADVANCED CYBER TRAINING PROGRAM

ICMPv6

ManTech

ICMP Definition

- The **Internet Control Message Protocol (ICMP)** is a helper protocol that supports IP with facility for:
  - Error reporting
  - Simple queries
- RFC 4443 outlines 4 error & 2 informational messages
- ICMP messages are encapsulated as IP datagrams:



IP header

ICMP message

IP payload

Compare to IPv4:
ICMPv4

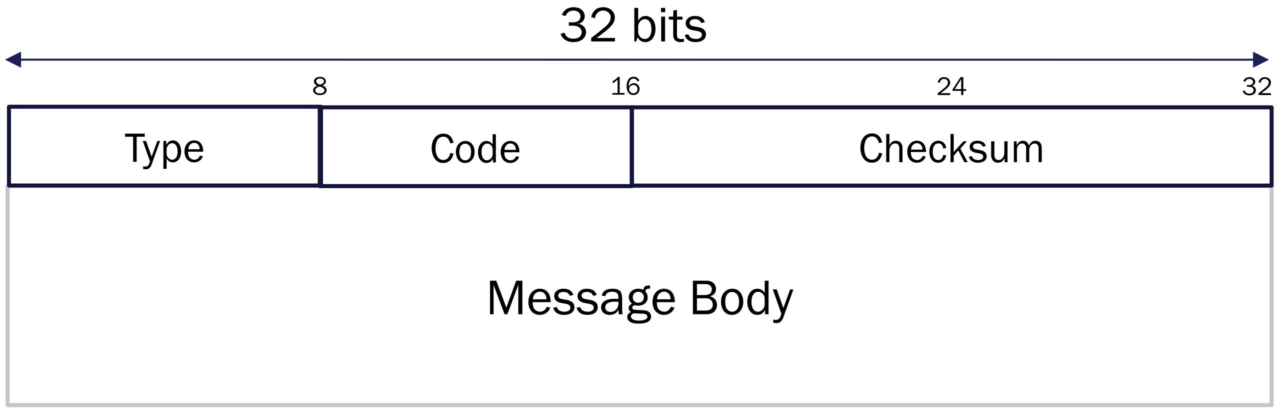
RFCs:
4443, 792 (ICMPv4)

ICMPv6 Mechanics

- IPv6 Next Header: 58
- ICMP Packet Format
  - Type (8 bits): the type of the message
  - Code (8 bits): additional level of message granularity
  - Checksum (16 bits): Detect Data Corruption

Compare to IPv4:
ICMPv6 requires a checksum

RFCs:
4443, 792 (ICMPv4)



32 bits

8	16	24	32

Type	Code	Checksum
Message Body		

ICMP Reporting and Message Types

Compare to IPv4:
Separation of Message Types

RFCs:
4443

- Internet Reporting (Error & Informational):
  - Error messages (type < 128) append parts of original packet up to the IPv6 MTU
  - Error messages of unknown type go to Upper Layer
  - Informational messages of unknown type will be silently discarded!
- ICMPv6 message type codes:

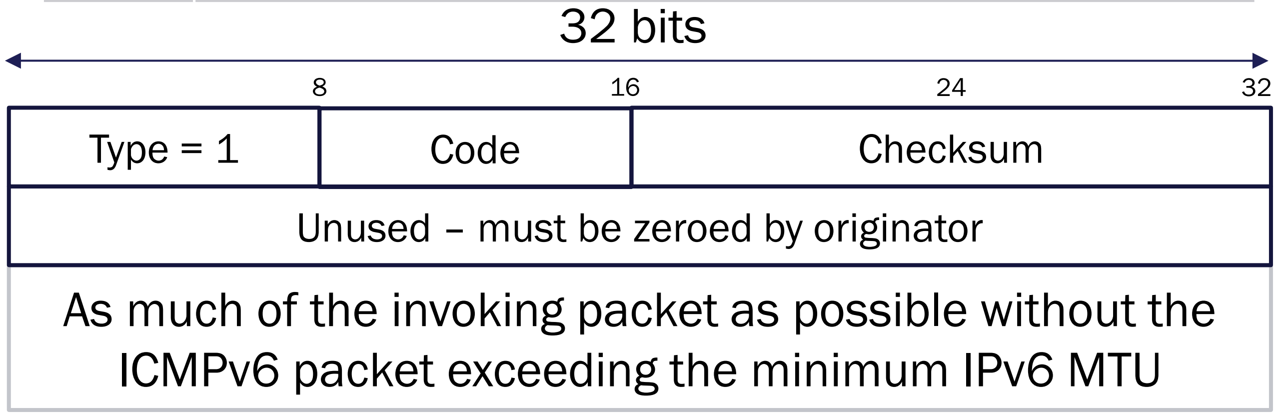
ICMPV6 ERROR MESSAGES	
1	Destination Unreachable
2	Packet too big
3	Time Exceeded
4	Parameter Problem

ICMPV6 INFORMATION MESSAGES	
128	Echo Request
129	Echo Reply

ICMPv6: Destination Unreachable

CODE	DESCRIPTION
0	No route to destination
1	Communication with destination administratively prohibited
2	Beyond scope of source address
3	Address unreachable
4	Port unreachable
5	Source address failed ingress/egress policy
6	Reject route to destination

RFCs:
4443

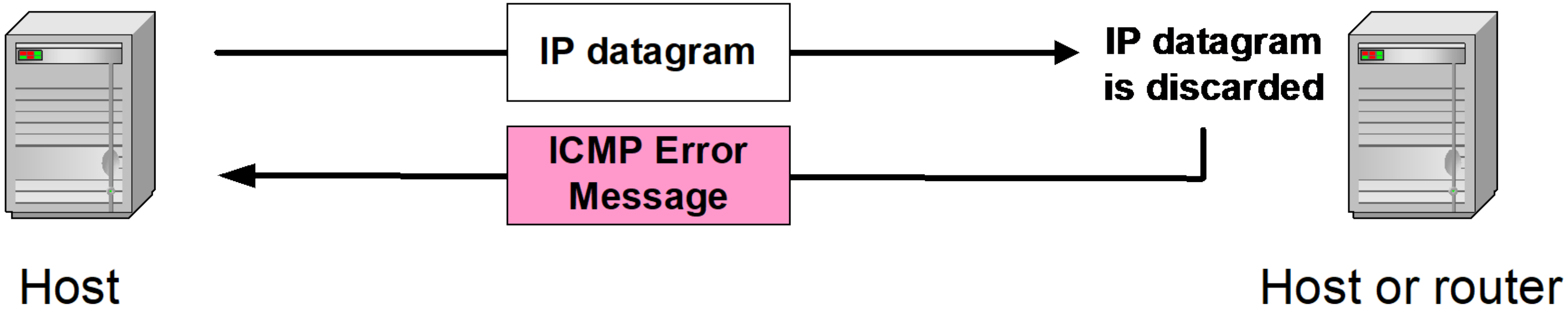

32 bits

8	16	24	32
Type = 1	Code	Checksum	
Unused – must be zeroed by originator			
As much of the invoking packet as possible without the ICMPv6 packet exceeding the minimum IPv6 MTU			

ICMP Error Message

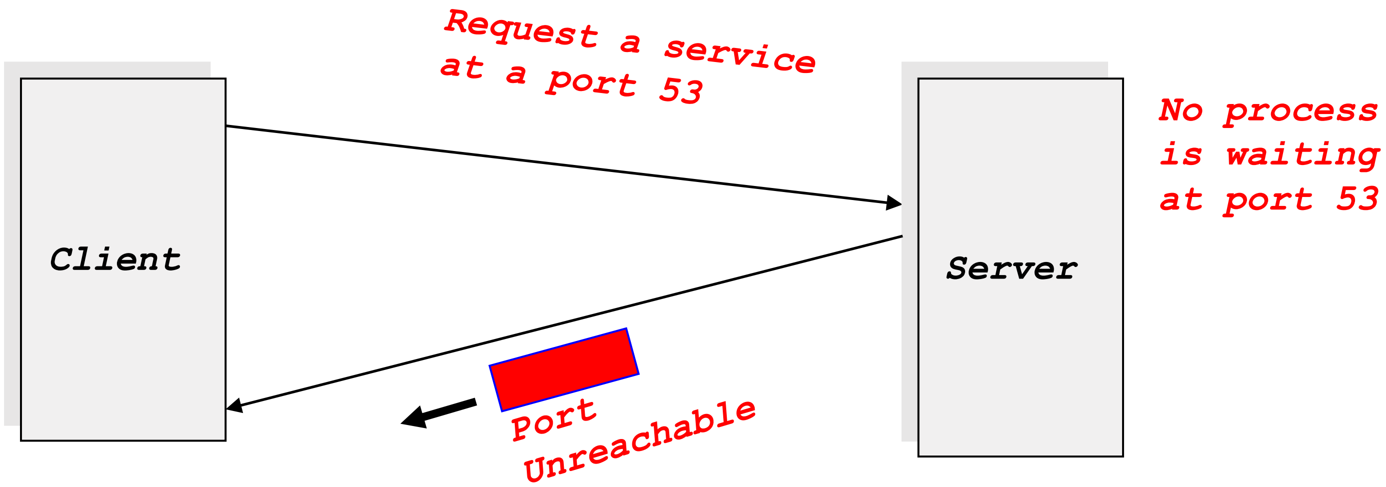
- ICMP error messages report error conditions
- Typically sent when a datagram is discarded
- Error message is often passed from ICMP to the application program

RFCs:
8200, 4443



Example: ICMP Port Unreachable

- STD 5: If, in the destination host, the IP module cannot deliver the datagram because the indicated protocol module or process port is not active, the destination host may send a destination unreachable message to the source host
- Scenario:



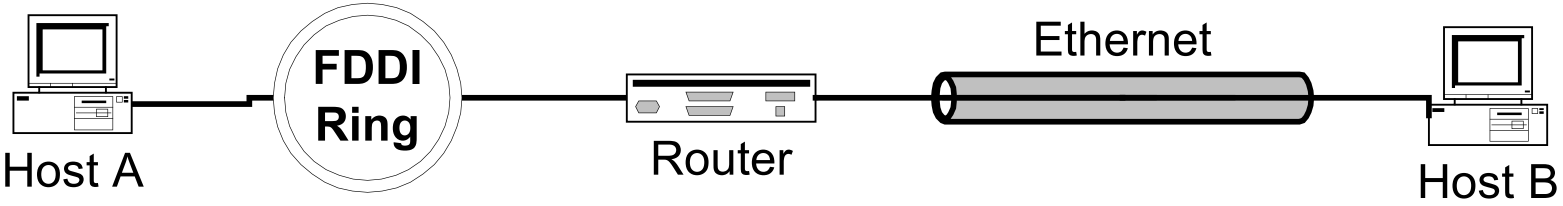
Maximum Transmission Unit

- Maximum size of IP datagram is 65535 (unless the jumbogram extended header is used)
- The data link layer protocol generally imposes a limit that is much smaller
- Example:
  - Ethernet frames have a maximum payload of 1500 bytes
  - Therefore, IP datagrams encapsulated in Ethernet frame cannot be longer than 1500 bytes
- The limit on the maximum IP datagram size, imposed by the data link protocol is called **maximum transmission unit (MTU)**

IP Fragmentation

- *What if the size of an IP datagram exceeds the MTU?*
  - IP datagram is fragmented into smaller units
- *What if the route contains networks with different MTUs?*
- Fragmentation
  - Source attempts Auto MTU discovery & fragments
  - Fragments are reassembled at receiver

RFCs:
8200, 4443

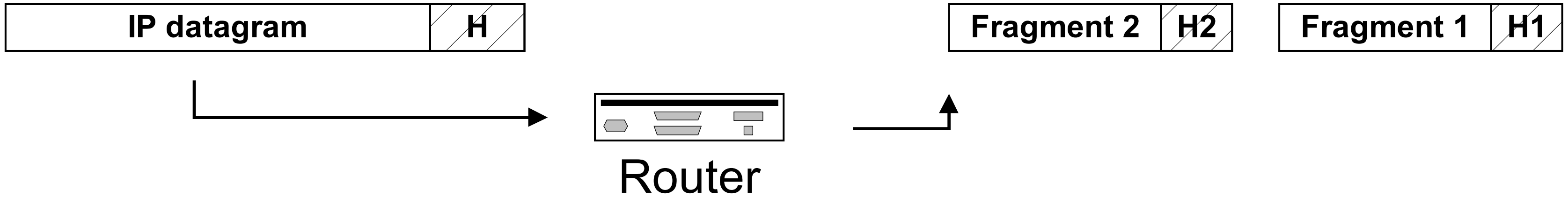


MTUs: FDDI: 4352 Ethernet: 1500

Where is Fragmentation done?

RFCs:
8200, 4443

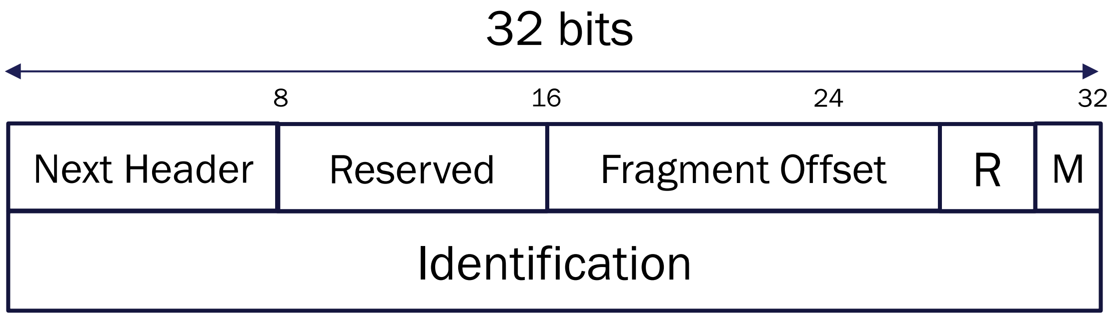
- Fragmentation can only be done at the sender
- Routers will discard packets that are larger than MTU and send back a “packet too big” ICMP error
- Reassembly of original datagram is done at destination host
- A sender will attempt to discover the smallest MTU on the path



How are Packets Fragmented?

RFCs:
8200, 4443

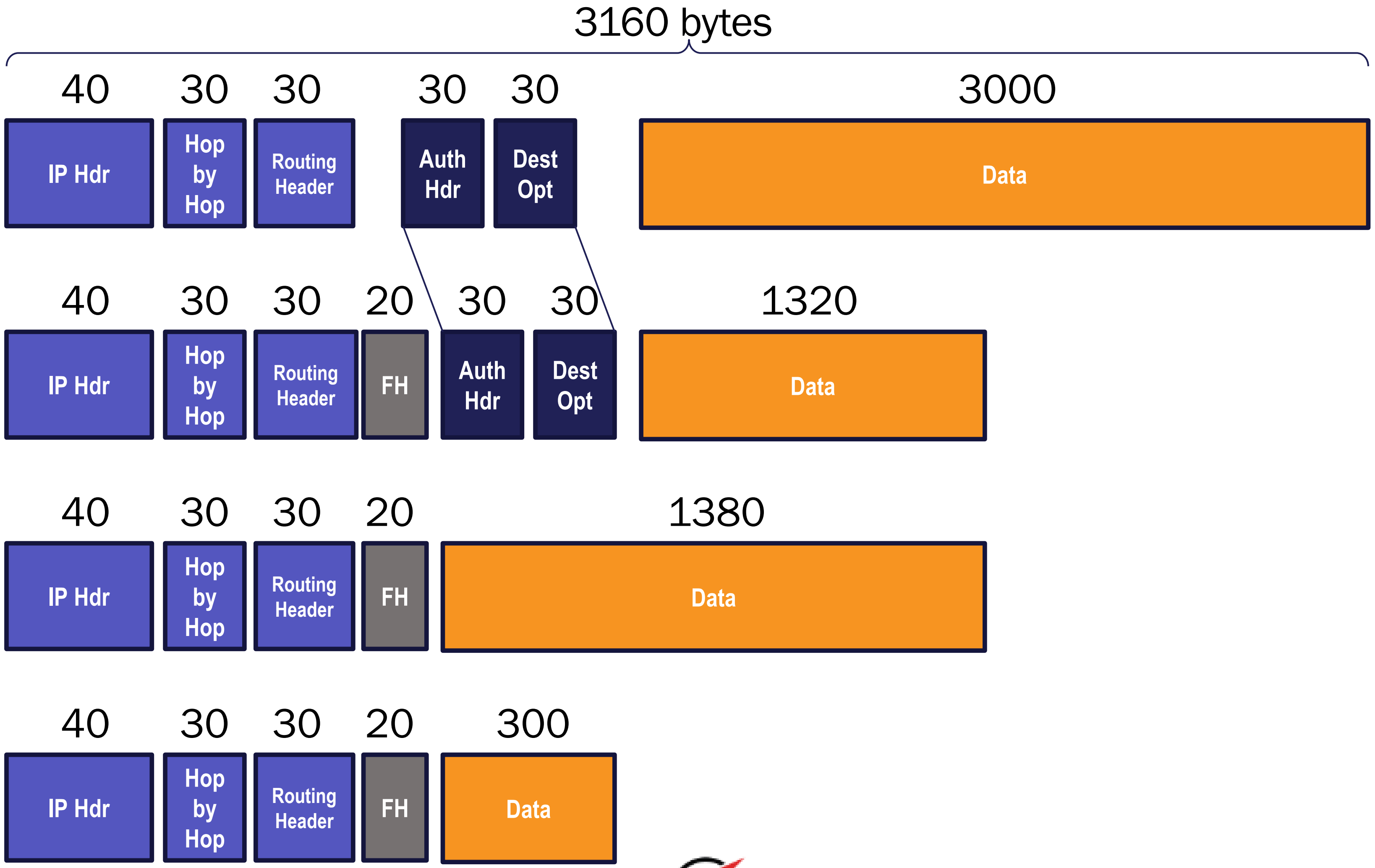
- Sender creates unique 32-bit identification
  - If routing header is present, the destination address is that of the final destination
- Unfragmentable part consists of IPv6 header and extension headers that must be processed by nodes en-route (routing header, hop-by-hop, destination opts)
- Fragmentable part consists of the rest
- Reassembled packet has all headers up to (and including the first) fragment header



Next Header	Reserved	Fragment Offset	R	M
Identification				

Packet Fragmentation Example

Let's assume a 1500 MTU



LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

Neighbor Discovery

Neighbor Discovery Protocol (NDP)

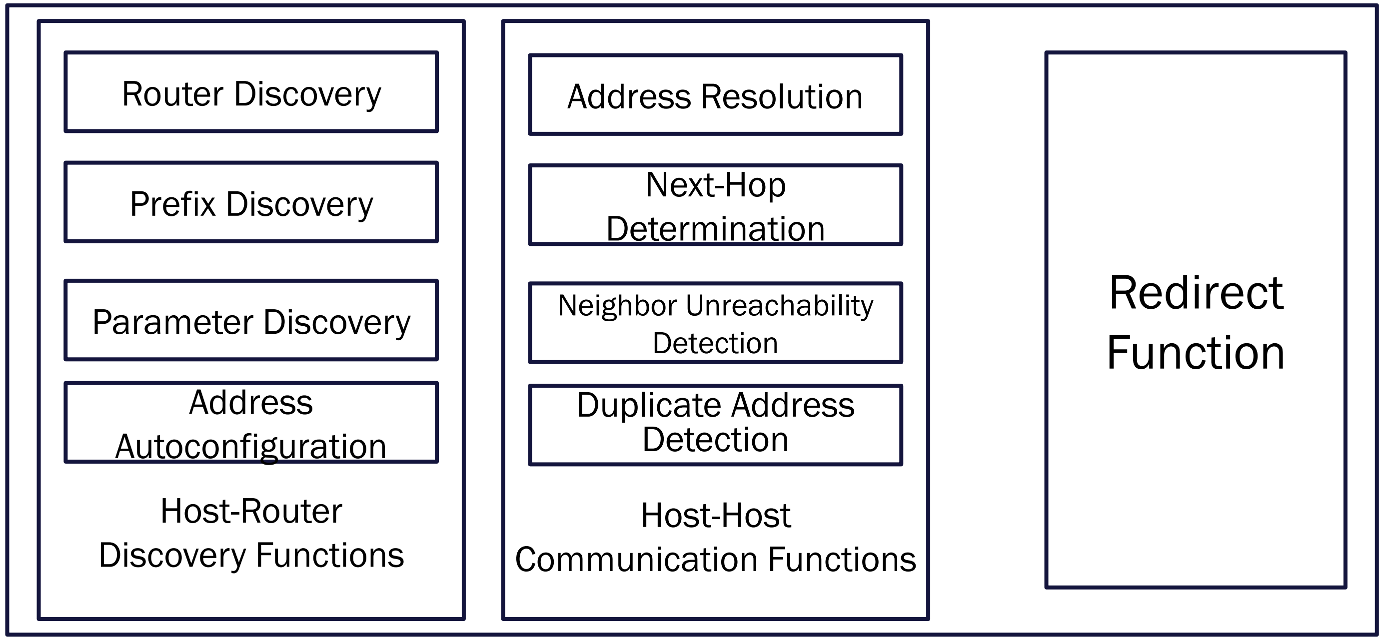
- Router Solicitation
- Router Advertisement
- Neighbor Solicitation
- Neighbor Advertising
- Redirect Message

Neighbor Discovery Protocol (NDP)

- Uses ICMPv6 (same next header number)
- Address auto-configuration
- Network Discovery/Awareness
- Miscellaneous Network Functions

Compare to IPv4:
None (New)

RFCs:
4861



Router Discovery

Prefix Discovery

Parameter Discovery

Address Autoconfiguration

Host-Router Discovery Functions

Address Resolution

Next-Hop Determination

Neighbor Unreachability Detection

Duplicate Address Detection

Host-Host Communication Functions

Redirect Function

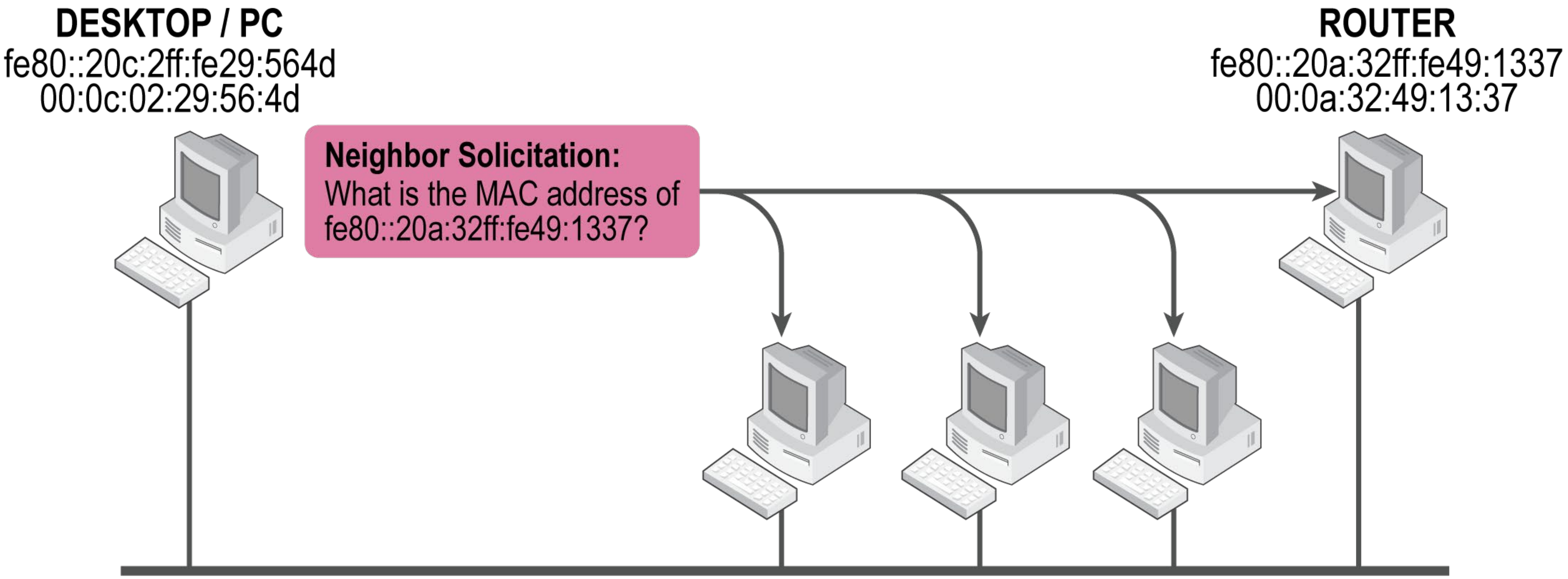
Address Resolution

- Neighbor Solicitations:

  “What is the hardware address of Router?”

Compare to IPv4:
Address Resolution Protocol

RFCs:
4861



DESKTOP / PC
fe80::20c:2ff:fe29:564d
00:0c:02:29:56:4d

Neighbor Solicitation:
What is the MAC address of fe80::20a:32ff:fe49:1337?

ROUTER
fe80::20a:32ff:fe49:1337
00:0a:32:49:13:37

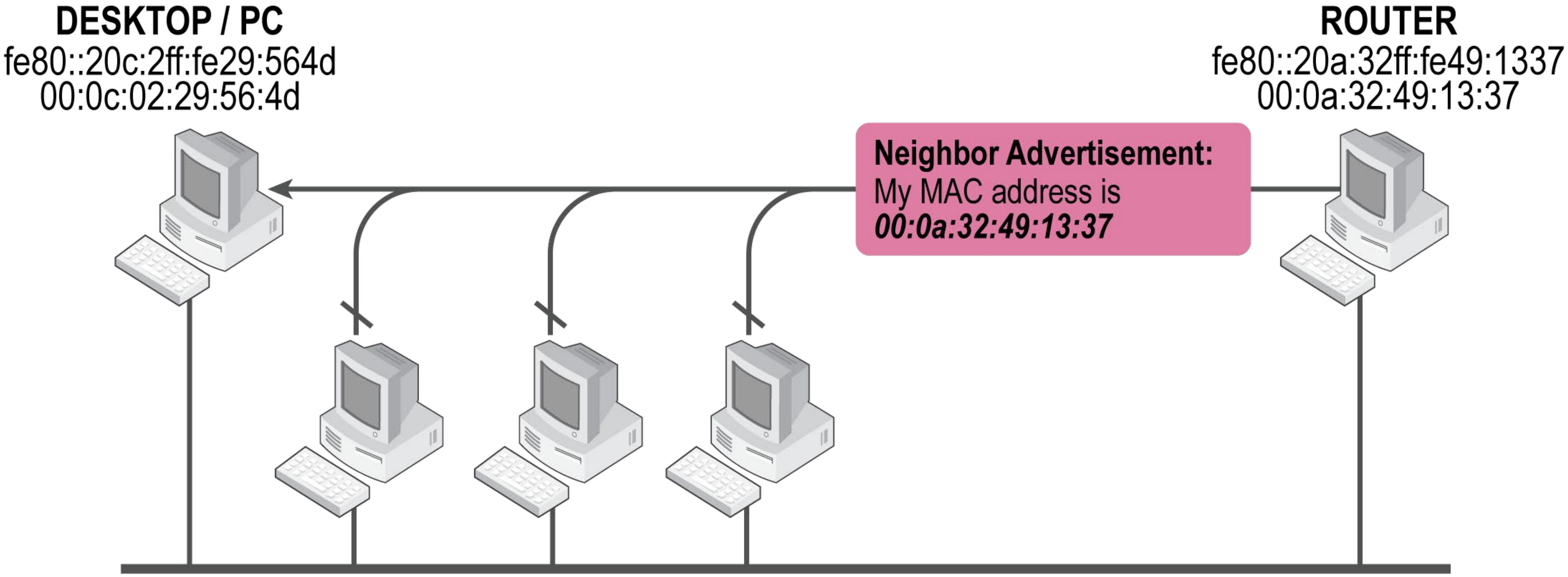
Address Resolution

- Neighbor Advertisement:

  "My link layer address is 00:0a:32:49:13:37"

Compare to IPv4:
Address Resolution Protocol

RFCs:
4861



Router Solicitation Message Format

- Type: 133
- Code: 0
- Checksum (ICMP Checksum)
- Reserved: Unused
- Options: Source link-layer address (if known)

Compare to IPv4:
None (New)

RFCs:
4861

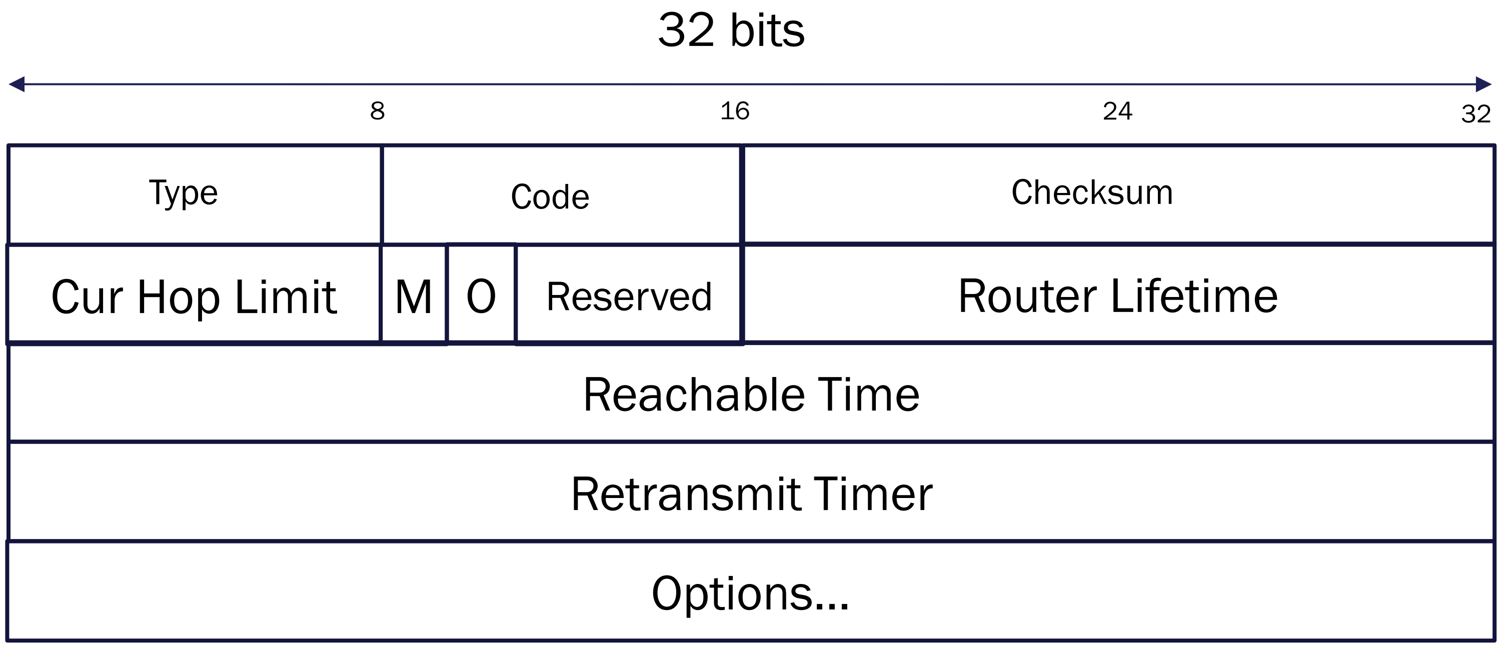
32 bits

<table>
<tr><td>8</td><td>16</td><td colspan="2">24   32</td></tr>
<tr><td>Type</td><td>Code</td><td colspan="2">Checksum</td></tr>
<tr><td colspan="4">Reserved</td></tr>
<tr><td colspan="4">Options...</td></tr>
</table>

Router Advertisement Message Format

Compare to IPv4:
None (New)

RFCs:
4861



<table>
<tr><td>Type</td><td colspan="3">Code</td><td>Checksum</td></tr>
<tr><td>Cur Hop Limit</td><td>M</td><td>O</td><td>Reserved</td><td>Router Lifetime</td></tr>
<tr><td colspan="5">Reachable Time</td></tr>
<tr><td colspan="5">Retransmit Timer</td></tr>
<tr><td colspan="5">Options...</td></tr>
</table>

Router Advertisement Format *(continued)*

Compare to IPv4:
None (New)

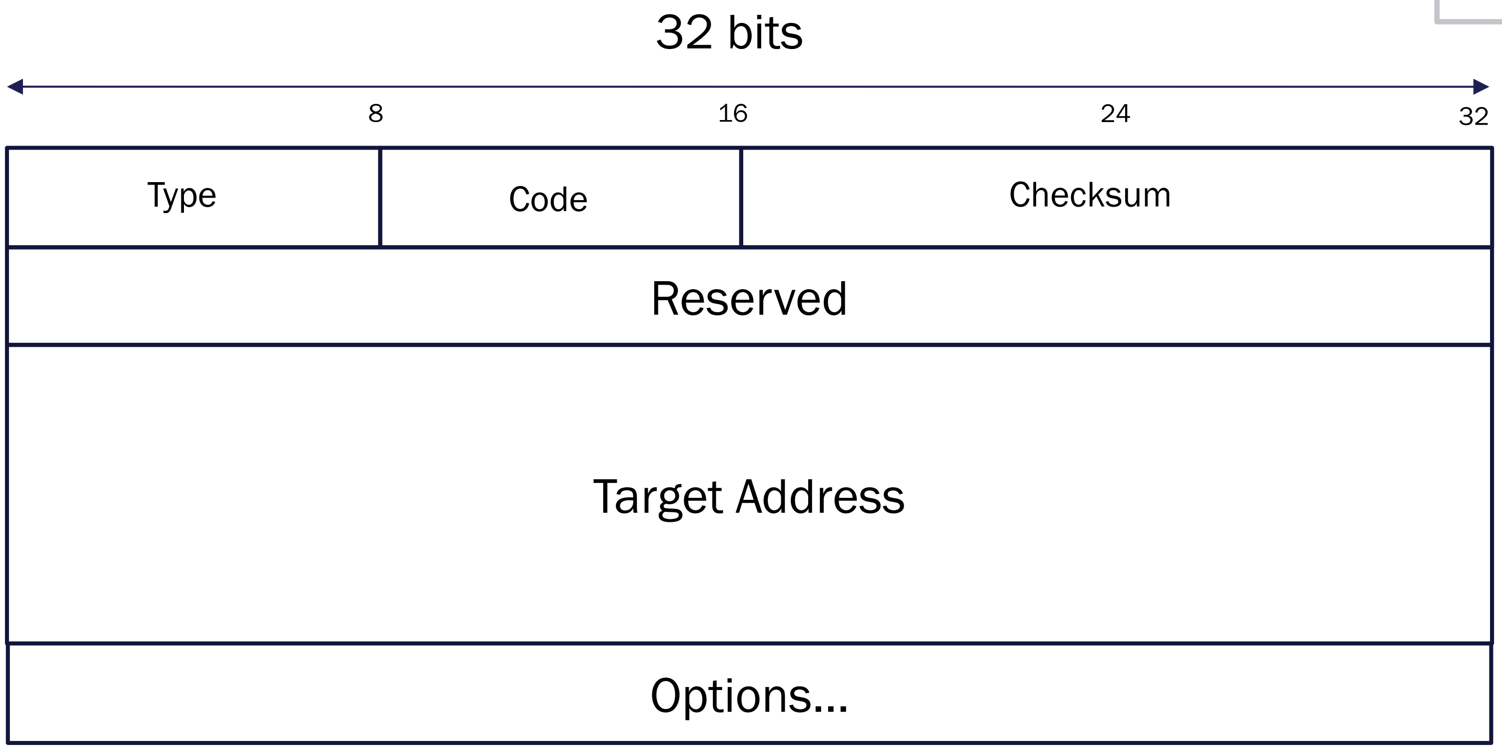
RFCs:
4861

- Type: 134
- Code: 0
- Checksum: ICMP checksum
- Cur Hop Limit (8-bit): default hop count value for outgoing packets
- M (1-bit): Managed address configuration flag
- O (1-bit): Other stateful configuration flag
- Reserved (6-bit): unused
- Router Lifetime (16-bit): Lifetime of default route
- Reachable Time (32-bit): Node “assumes” neighbor is present
- Retransmission Timer (32-bit): time between retransmissions
- Options: Source link-layer address, MTU, prefix information

Neighbor Solicitation Message Format

Compare to IPv4:
None (New)

RFCs:
4861



Type	Code	Checksum
Reserved		
Target Address		
Options...		

Neighbor Solicitation Format *(continued)*

- Type: 135
- Code: 0
- Checksum: ICMP checksum
- Reserved (32-bit): unused (all zeros)
- Target Address (128-bit): IP address of target of solicitation (cannot be a multicast address)
- Options: Source link-layer address

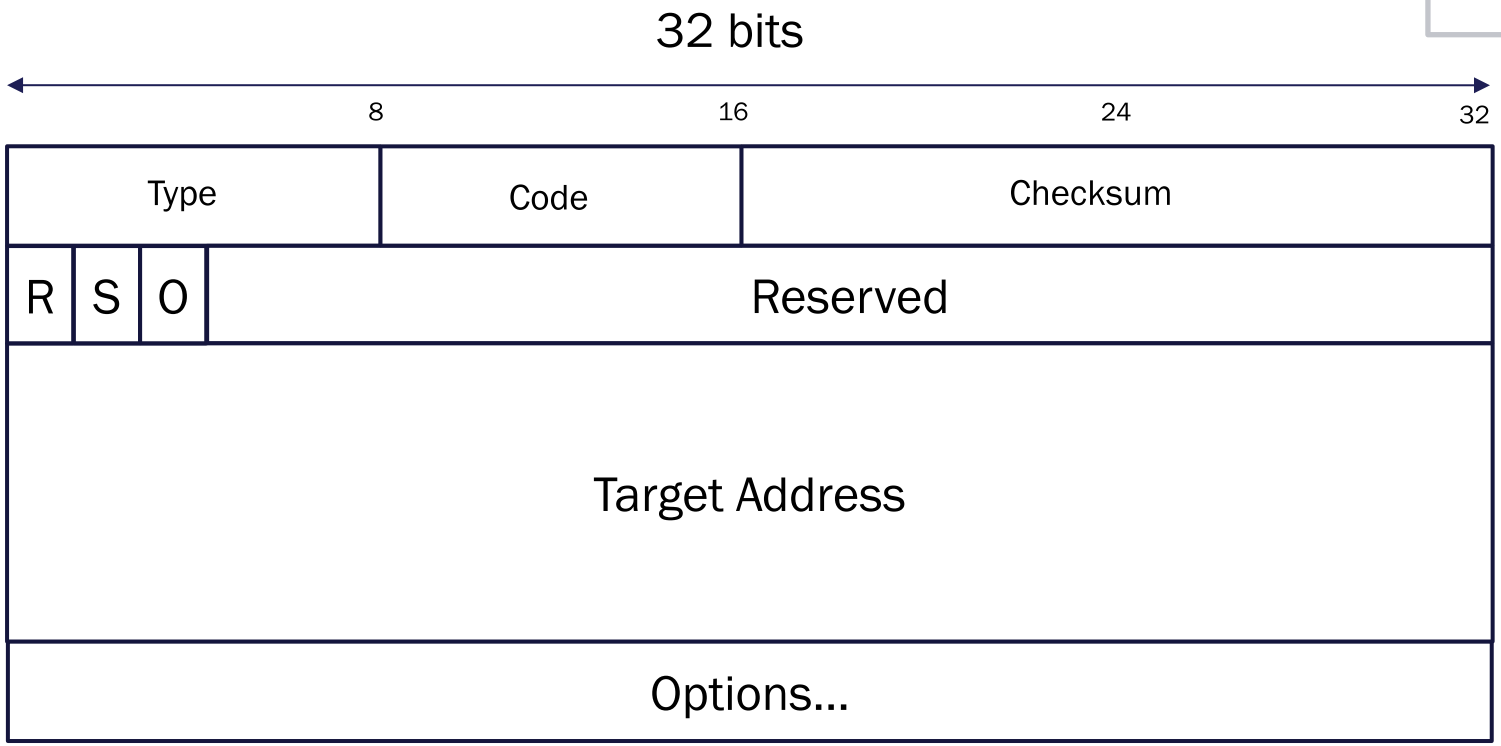
Compare to IPv4:
None (New)

RFCs:
4861

Neighbor Advertisement Message Format

Compare to IPv4:
None (New)

RFCs:
4861



Neighbor Advertisement Format *(continued)*

Compare to IPv4:
None (New)

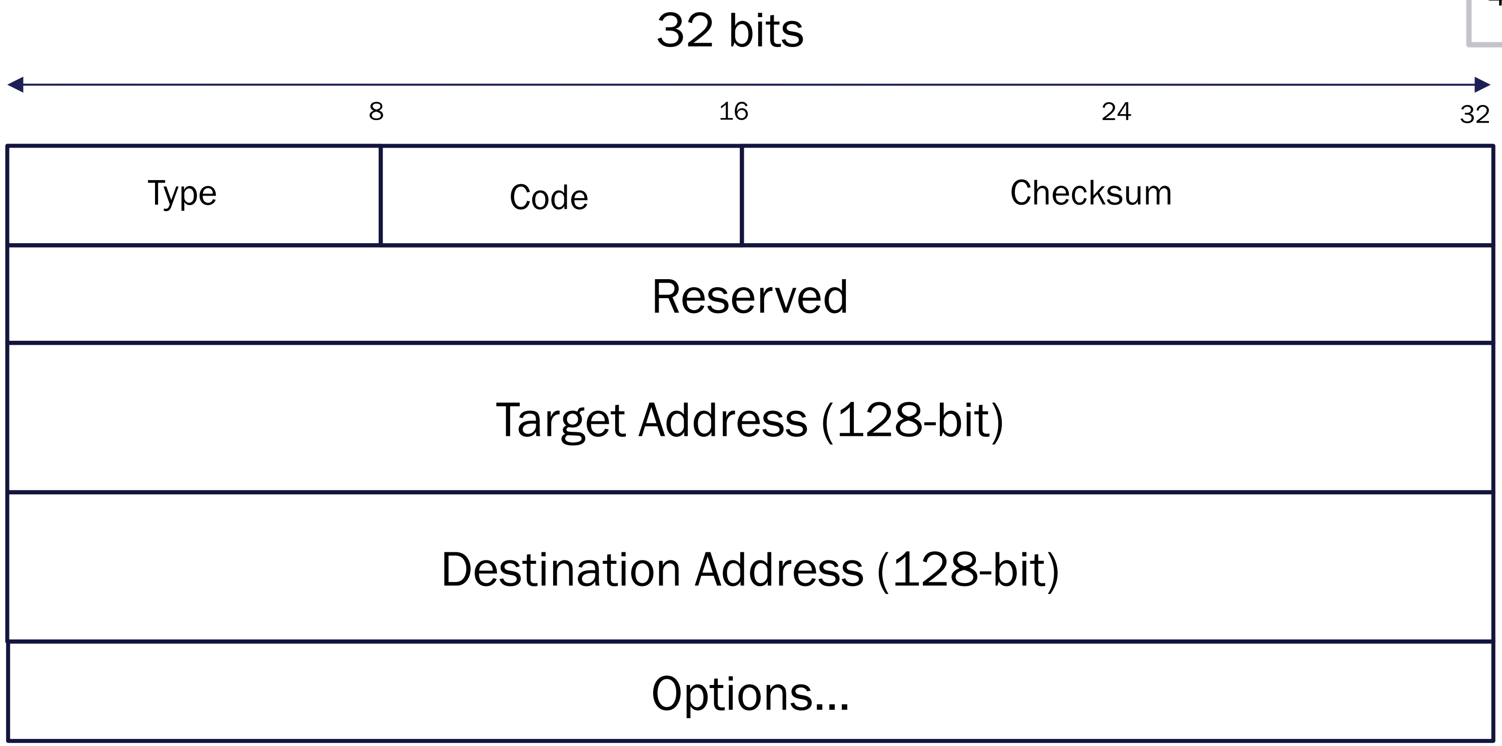
RFCs:
4861

- Type (8-bit): 136
- Code (8-bit): 0
- Checksum (16-bit): ICMP checksum
- R (1-bit): Router flag – indicates sender is a router
- S (1-bit): Solicited flag – indicates response to solicitation
- O (1-bit): Override flag – indicates response should override existing cache entries
- Reserved (29-bit): unused (all zeros)
- Target Address (128-bit): depends (solicited or not solicited)
- Options: target link-layer address (sender of advertisement)

Redirect Message Format

Compare to IPv4:
None (New)

RFCs:
4861



<table>
<tr><td>Type</td><td>Code</td><td>Checksum</td></tr>
<tr><td colspan="3">Reserved</td></tr>
<tr><td colspan="3">Target Address (128-bit)</td></tr>
<tr><td colspan="3">Destination Address (128-bit)</td></tr>
<tr><td colspan="3">Options…</td></tr>
</table>

Redirect Message Format *(continued)*

Compare to IPv4:
None (New)

RFCs:
4861

- Type (8-bit): 137
- Code (8-bit): 0
- Checksum (16-bit): ICMP checksum
- Reserved (32-bit): unused (all zeros)
- Target Address (128-bit): IP address that is a better first hop to use for ICMP Destination address
- Destination Address (128-bit): IP address of destination that is redirected through target
- Options: target link-layer address, redirected header (as much as possible of the IP packet that triggered the sending of the redirect without fragmenting the packet)

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

Lab 2

IPv6 & ICMPv6

ManTech

Lab Intro: Clarifications

- Your “a:c:7:9::” address is the “Global” one, and the address you should use for tasks 1 & 2. Your “fe80::” address is “Link-Local” and should only be used in task 3.
- IPv6 does not use ARP. It uses NDP instead

Overview

- **TASK 0:** RFC navigating
- **TASK 1:** Neighbor Solicitations
- **TASK 2:** Neighbor Advertisements
- **TASK 3:** Router Advertisements

TASK 0: RFC Navigating

1. Check out the RFC index
2. Obseleted, updated by, obseletes, etc

TASK 1: DIY Neighbor Solicit

- Construct a Neighbor Solicitation frame
- Starter/Hint code:

```
>>> dst = b"\x33\x33\x00\x00\x00\x01"  #(IPv6Mcast)
>>> src = b"\x??" * 6
>>> eth_type = b"\x86\xdd"        # (Type: IPv6)
>>> etherhdr = dst + src + eth_type
```

TASK 1: DIY Neighbor Solicit

(continued)

```
>>> ipv6hdr= b"\x60\x00\x00\x00"# version 6, traffic class and flow label
>>> ipv6hdr += b"\x??\x??"      # payload len (2 bytes)
>>> ipv6hdr += b"\x3a"    # next header
>>> ipv6hdr += b"\xff"    # hop limit
>>> ipv6hdr += b"\x??" * 16     # src ip
>>> ipv6hdr += b"\x??" * 16     # dst ip
```

TASK 1: DIY Neighbor Solicit *(continued)*

- ICMP v6 Header

```
>>> icmpv6hdr  = b"\x87”  #type (neighbor
solicitation)
>>> icmpv6hdr += b"\x00”  #code: 0
>>> icmpv6hdr += b"\x00\x00”    #checksum
>>> icmpv6hdr += b"\x00” * 4    #reserved
>>> icmpv6hdr += b"\x??” * 16   #target ip
>>> icmpv6hdr += icmpv6option
```

TASK 1: DIY Neighbor Solicit *(continued)*

- ICMP v6 Header

```
>>> icmpv6option = b"\x01"       # type 1 = source
        # link layer address
>>> icmpv6option += b"\x01"      # length = 8
>>> icmpv6option += b"\x??\x??\x??\x??\x??\x??"
        # link layer addr
```

TASK 1: DIY Neighbor Solicit

(continued)

- For best results, read the RFC! (RFC 4861)
- Make sure your sniffer is running
- Using provided helper functions, send your frame

```
>>> rawsend_cksum_ipv6(<your entire frame here>, dev="ens33")
```

- Did the remote host reply to your request?
  - Use Wireshark to verify
- What is the difference between the destination IP and target IP?

TASK 2: DIY Neighbor Advertisement

- For best results, read the RFC! (RFC 4861)
- Make sure your sniffer is running
- Using provided helper functions, send your frame

```
>>> rawsend_cksum_ipv6(<your entire frame here>, dev="ens33")
```

- Those flags look interesting, what are they used for? (Read the RFC)

TASK 3: Router Advertisement

- That Router flag looks fun…
- Use a copy of your previously working code
- Modify it so that you have a Router Advertisement!
- You should be able to force your neighbors to auto configure themselves with an arbitrary prefix. (B:C:7:9::? C:C:7:9::?)
- Try stuff, break stuff, we can always reboot ;-)

(Don’t plan on having Five Nines uptime…)

Neighbor Discovery Protocol

- Validation of Neighbor Solicitations (RFC 4861 – 7.1.1)
- “A node MUST silently discard any received Neighbor Solicitation messages that do not satisfy all of the following validity checks”
- What other restrictions are mentioned in the RFC?
- How do they affect your code?

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

Transport Layer

ACTP
ADVANCED CYBER TRAINING PROGRAM

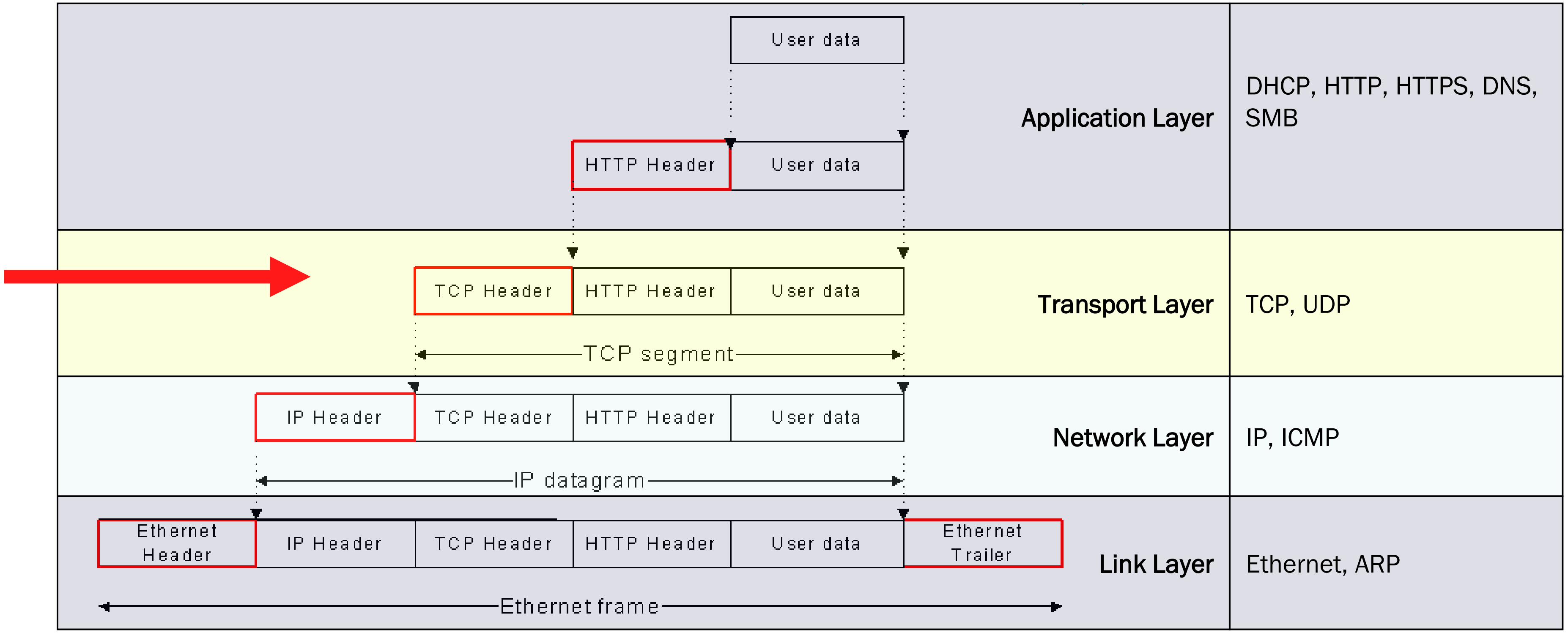
Objectives

Given a workstation, device, and/or technical documentation, the student will be able to:

- Learning Objective
  - Use common Transport Layer protocol specifications to analyze, interpret, and construct network traffic
- Enabling Objectives
  - Describe UDP in terms of its fields and how it functions
  - Describe TCP in terms of its fields and how it functions
  - Describe flow control and reliability in relation to TCP
  - Describe the differences between transport layer protocols
  - Construct a simple TCP packet

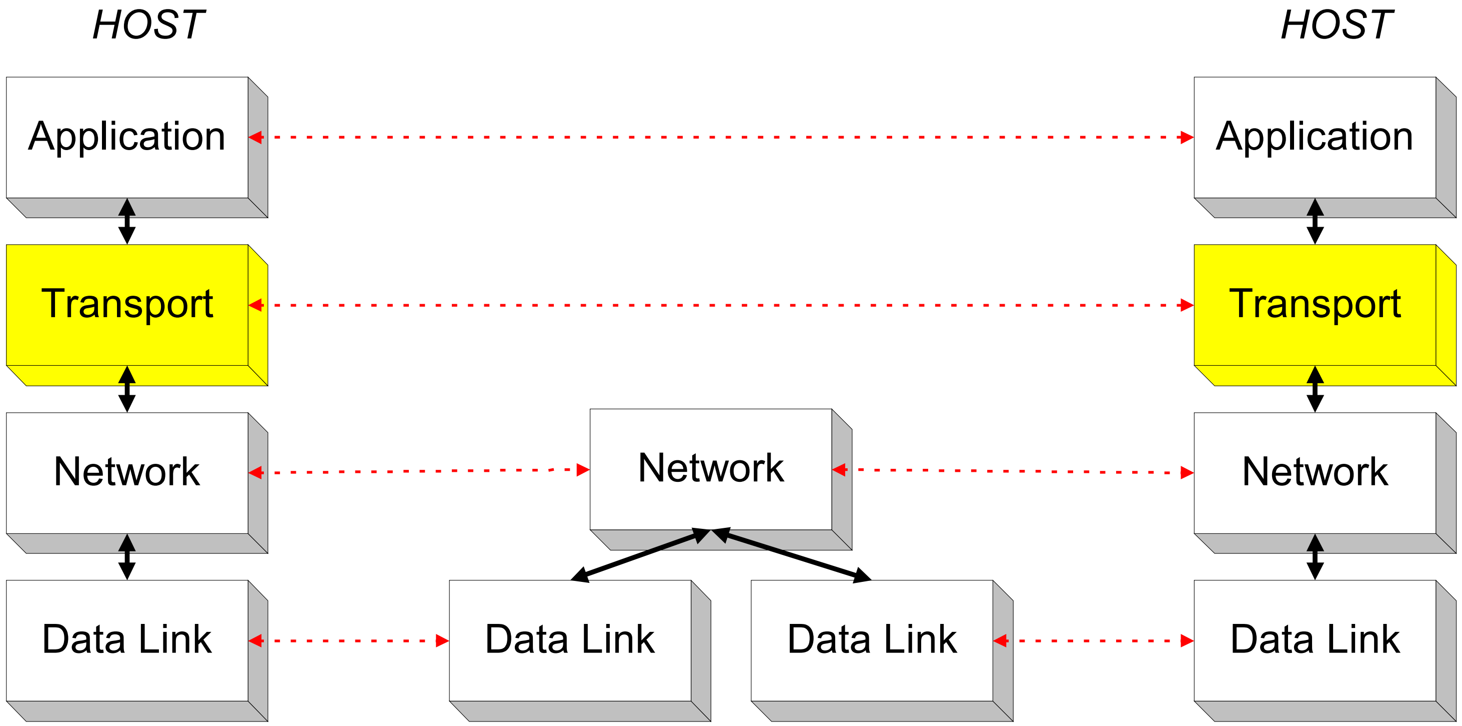
Transport Layer

- The Transport layer involves the mechanisms used for data exchange with regards to software



Orientation

- Transport layer protocols are end-to-end protocols
- They are only implemented at the hosts



Transport Protocols Over The Internet

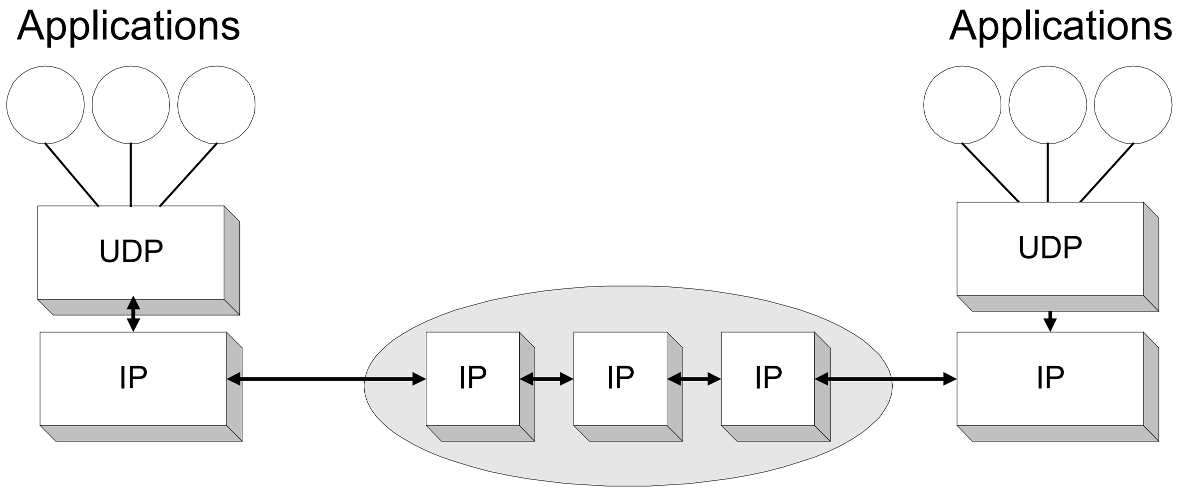
RFCs:
768, 9293

- TCP – Transmission Control Protocol
    - stream oriented
    - reliable, connection-oriented
    - complex
    - only unicast
    - used for most Internet applications:
        - web (HTTP), email (SMTP), file transfer (FTP), terminal (telnet), secure shell (SSH), large DNS
- UDP - User Datagram Protocol
    - datagram oriented
    - unreliable, connectionless
    - simple
    - unicast or multicast
    - useful only for few applications, e.g., multimedia applications
    - used a lot for services
        - network management (SNMP), routing (RIP), naming (DNS)

User Datagram Protocol (UDP)

RFCs:
768

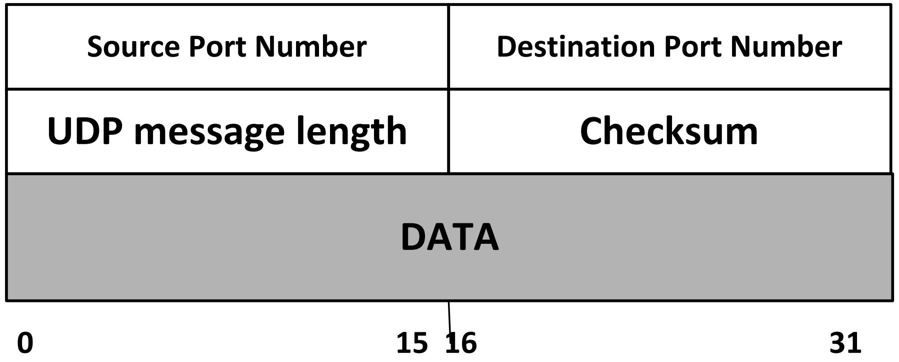
- UDP supports unreliable transmissions of datagrams
- UDP merely extends the host-to-to-host delivery service of IP datagram to an application-to-application service
- The only thing that UDP adds is multiplexing and demultiplexing



UDP Format

RFCs:
768

- **Port numbers** identify sending and receiving applications (processes); maximum port number is 2^16-1= 65,535
- **Message Length** is at least 8 bytes (i.e., Data field can be empty) and at most 65,535
- **Checksum** is for header (of UDP and some of the IP header fields, see pseudo headers) and Data

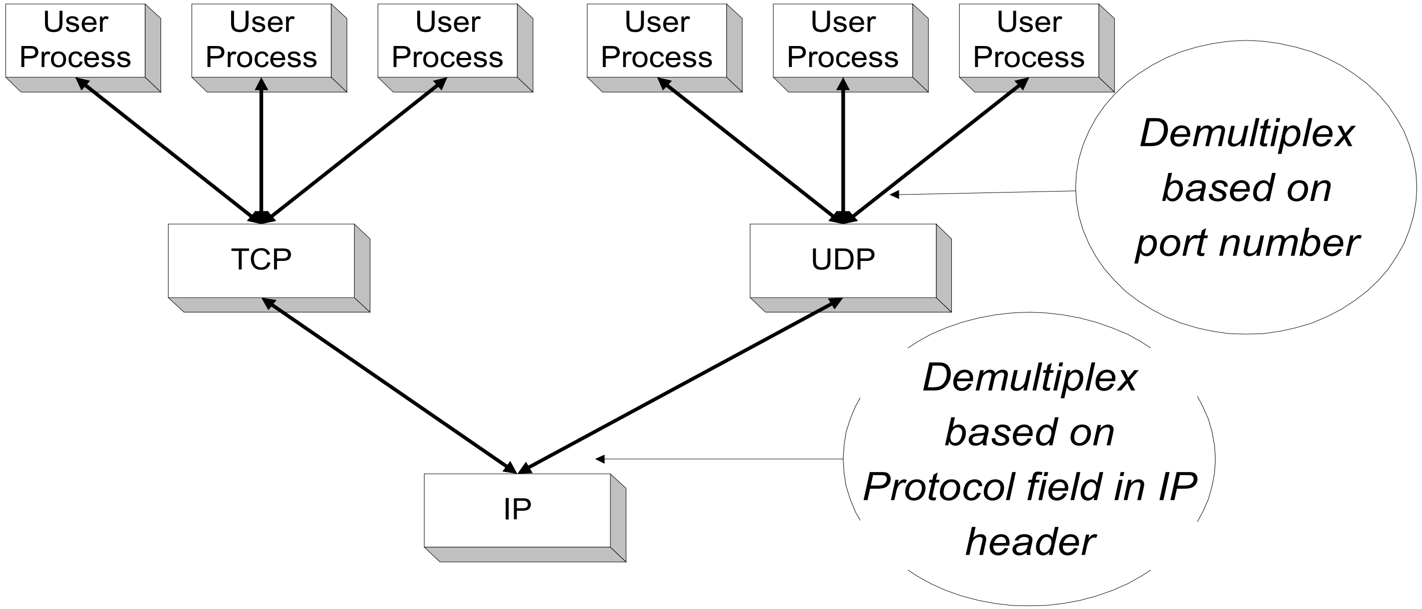


Source Port Number	**Destination Port Number**
UDP message length	**Checksum**
DATA	

Port Numbers

- UDP (and TCP) use port numbers to identify applications
- A globally unique address at the transport layer (for both UDP and TCP) is a tuple <IP address, port number>
- There are 65,535 UDP/TCP ports per host

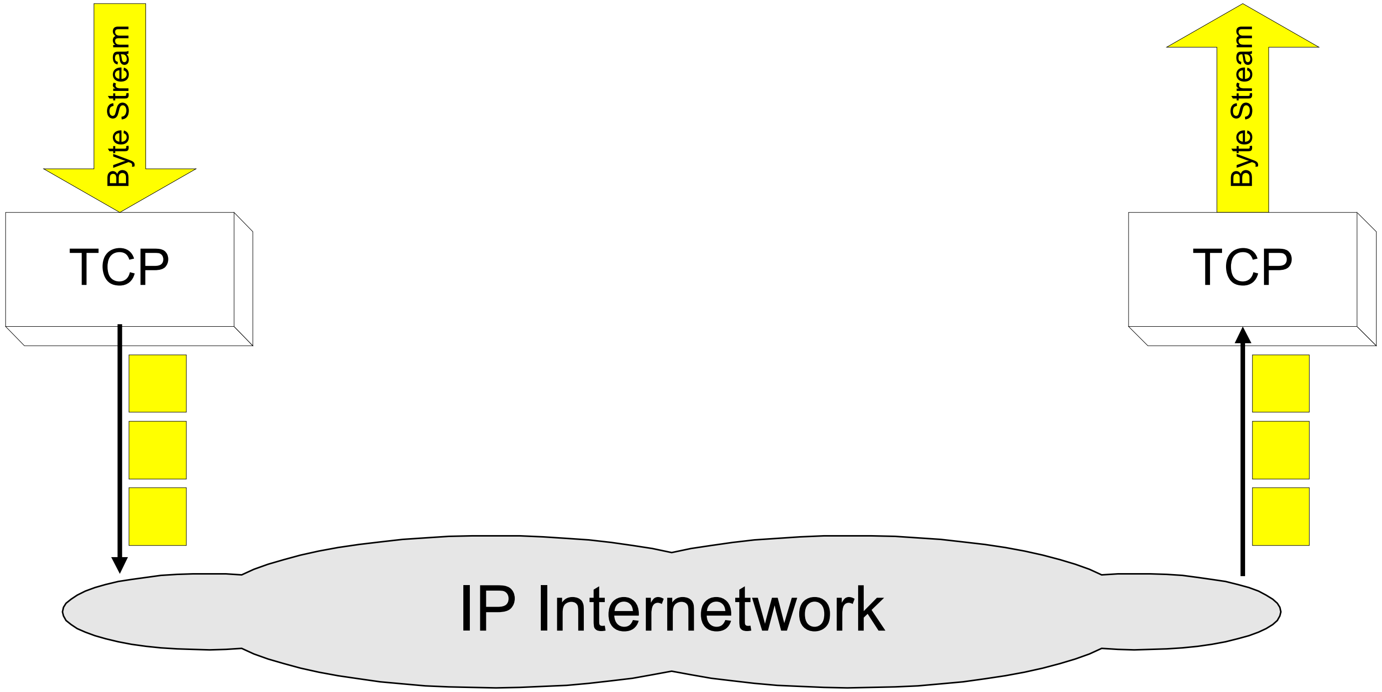
RFCs:
768, 9293



Transmission Control Protocol (TCP)

- TCP is a connection-oriented protocol that provides a reliable unicast end-to-end byte stream over an unreliable internetwork.

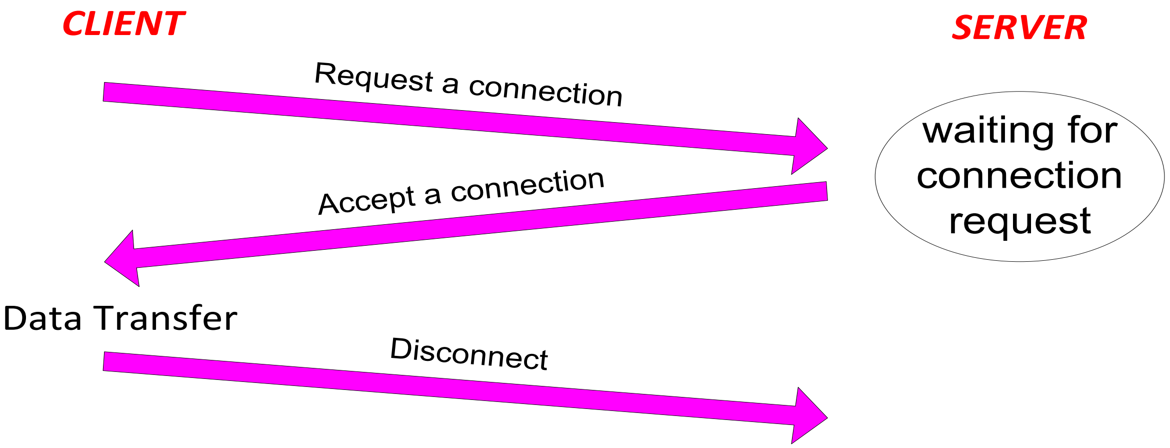
RFCs:
9293



Connection Oriented

- Before any data transfer, TCP establishes a connection
  - One TCP entity is waiting for a connection (“server”)
  - The other TCP entity (“client”) contacts the server
- The actual procedure for setting up connections is more complex
- Each connection is full duplex

RFCs:
9293



Reliable

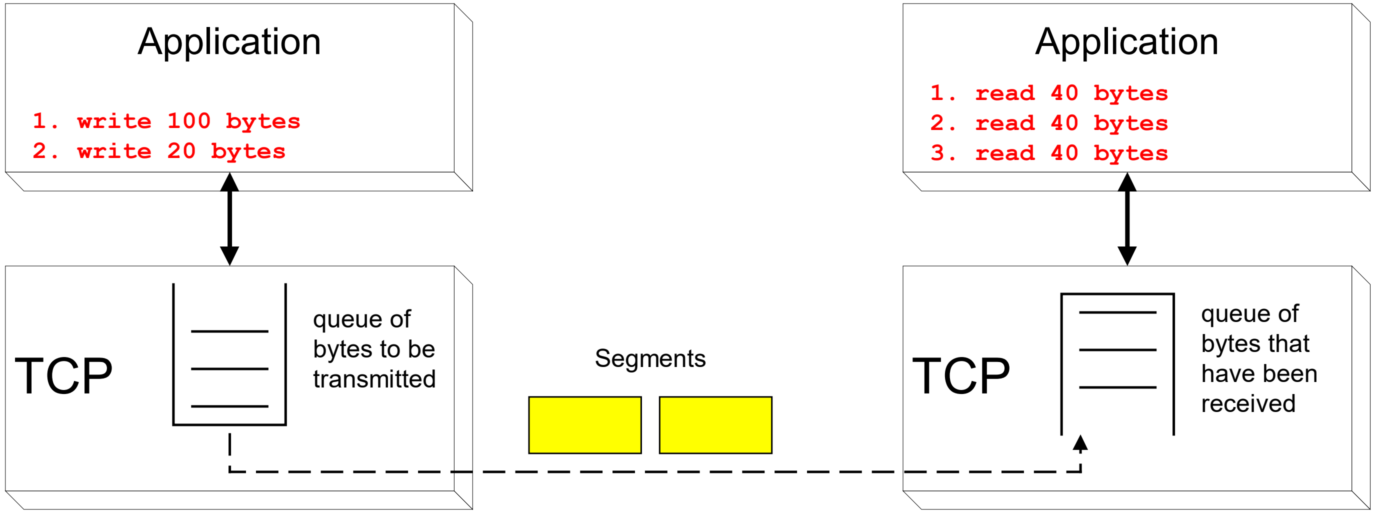
RFCs:
9293

- Byte stream is broken up into chunks which are called segments
  - Receiver sends acknowledgements (ACKs) for segments
  - TCP maintains a timer. If an ACK is not received in time, the segment is retransmitted
- Detecting errors
  - TCP has checksums for header and data
    - Segments with invalid checksums are discarded
  - Each byte that is transmitted has a sequence number
    - SYN and FIN flags also have SeqNo’s, why?

Byte Stream Service

RFCs:
9293

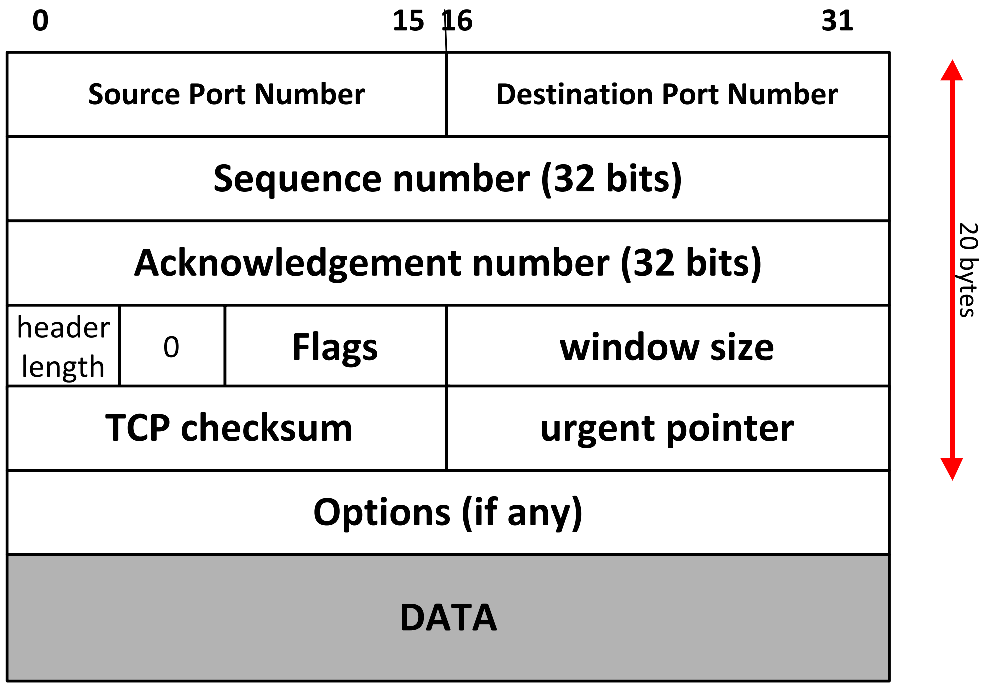
- To the lower layers, TCP handles data in blocks, the segments
- To the higher layers TCP handles data as a sequence of bytes and does not identify boundaries between bytes
- So: Higher layers do not know about the beginning and end of segments!



TCP Format

- TCP has a minimum 20 byte header followed by ≥ 0 bytes of data

RFCs:
9293



Source Port Number			Destination Port Number
Sequence number (32 bits)			
Acknowledgement number (32 bits)			
header length	0	Flags	window size
TCP checksum			urgent pointer
Options (if any)			
DATA			

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

TCP Header Fields

Port Number

RFCs:
9293

- A port number identifies the endpoint of a connection
- A pair **(IP address, port number)** identifies one endpoint of a connection
- Two pairs **(client IP address, client port number)** and **(server IP address, server port number)** identify a TCP connection

Sequence Number (SeqNo)

RFCs:
9293

- Sequence number in TCP is 32 bits long
- The range is

  $0 \leq \text{Sequence number} \leq 2^{32} -1 \approx 4.3 \text{ Gbyte}$
- Each sequence number identifies a byte in the byte stream
- The Initial Sequence Number (ISN) is the initial value for the sequence number
- The client and the server each select the ISN randomly during connection establishment

Acknowledgement Number (AckNo)

RFCs:
9293

- Acknowledgements are piggybacked
  - A segment from A-to-B can contain an acknowledgement for a data sent in the B-to-A direction
- A hosts uses the AckNo field to send acknowledgements
  - (If a host sends an AckNo in a segment it sets the “ACK flag”)
- The AckNo contains the next SeqNo that a hosts wants to receive
- Example:
  - The acknowledgement for a segment with sequence numbers 0-1500 is AckNo=1501

Acknowledgement Number *(continued)*

RFCs:
9293

- TCP uses the sliding window flow protocol to regulate the flow of traffic from the sender to receiver
- TCP uses the following variation of the sliding window protocol:
    - no NACKs (Negative ACKnowledgement)
    - only cumulative ACKs
- Example:
    - Assume: Sender sends two segments with "1..1500" and "1501..3000", but receiver only gets the second segment
    - In this case, the receiver cannot acknowledge the second packet; it can only send AckNo=1

Data Offset (Header Length)

- 4 bits
- Length of header in 32-bit words
- Note that TCP header has variable length with minimum 20 bytes

RFCs:
9293

Flag Bits

RFCs:
9293

- **CWR:** Congestion Window Reduced
- **ECE:** ECN-Echo
  - During the synchronization phase of a connection between client and server, the TCP CWR and ECE flags work in conjunction to establish whether the connection is capable of leveraging congestion notification. In order to work, both client and server need to support ECN.
- **URG:** Urgent pointer is valid
  - If the bit is set, the following bytes contain an urgent message in the range:

    SeqNo ≤ urgent message ≤ SeqNo + urgent pointer
- **ACK:** Acknowledgment Number is valid
- **PSH:** PUSH Flag
  - Notification from sender to the receiver that the receiver should pass all data that it has to the application
  - Normally set by sender when the sender’s buffer is empty

Flag Bits *(continued)*

- **RST:** Reset the connection
  - The flag causes the receiver to reset the connection
  - Receiver of a RST terminates the connection and indicates higher layer application about the reset
- **SYN:** Synchronize sequence numbers
  - Sent in the first packet when initiating a connection
- **FIN:** Sender is finished with sending
  - Used for closing a connection
  - Both sides of a connection must send a FIN

RFCs:
9293

Window Size

- Each side of the connection advertises the window size
- Window size is the maximum number of bytes that a receiver can accept
    - 3 parts of the algorithm. Stop and Wait, Go Back N, Selective Repeat
- Maximum window size is 2^16-1= 65535 bytes
- “Correct” size determined by throughput & latency (BDP)

RFCs:
9293

TCP Checksum and Urgent Pointer

- TCP Checksum:
  - TCP checksum covers both TCP header and TCP data (also covers some parts of the IP header)
- Urgent Pointer:
  - Only valid if URG flag is set

RFCs:
9293

Options

RFCs:
9293, 1323

End of Options

kind=0	
1 byte	

NOP (no operation)

kind=1	
1 byte	

Maximum Segment Size

kind=2	len=4	maximum segment size
1 byte	1 byte	2 bytes

Window Scale Factor

kind=3	len=3	shift count
1 byte	1 byte	1 byte

Timestamp

kind=8	len=10	timestamp value	timestamp echo reply
1 byte	1 byte	4 bytes	4 bytes

Options *(continued)*

RFCs:
9293

- **NOP** is used to pad TCP header to multiples of 4 bytes
- **Maximum Segment Size**
- **Window Scale Options**
  - Increases the TCP window from 16 to 32 bits, I.e., the window size is interpreted differently

  What is the different interpretation?
  - This option can only be used in the SYN segment (first segment) during connection establishment time
- **Timestamp Option**
  - Can be used for roundtrip measurements

TCP Connection Management

- Opening a TCP Connection
- Closing a TCP Connection
- Special Scenarios
- State Diagram

TCP Connection Establishment

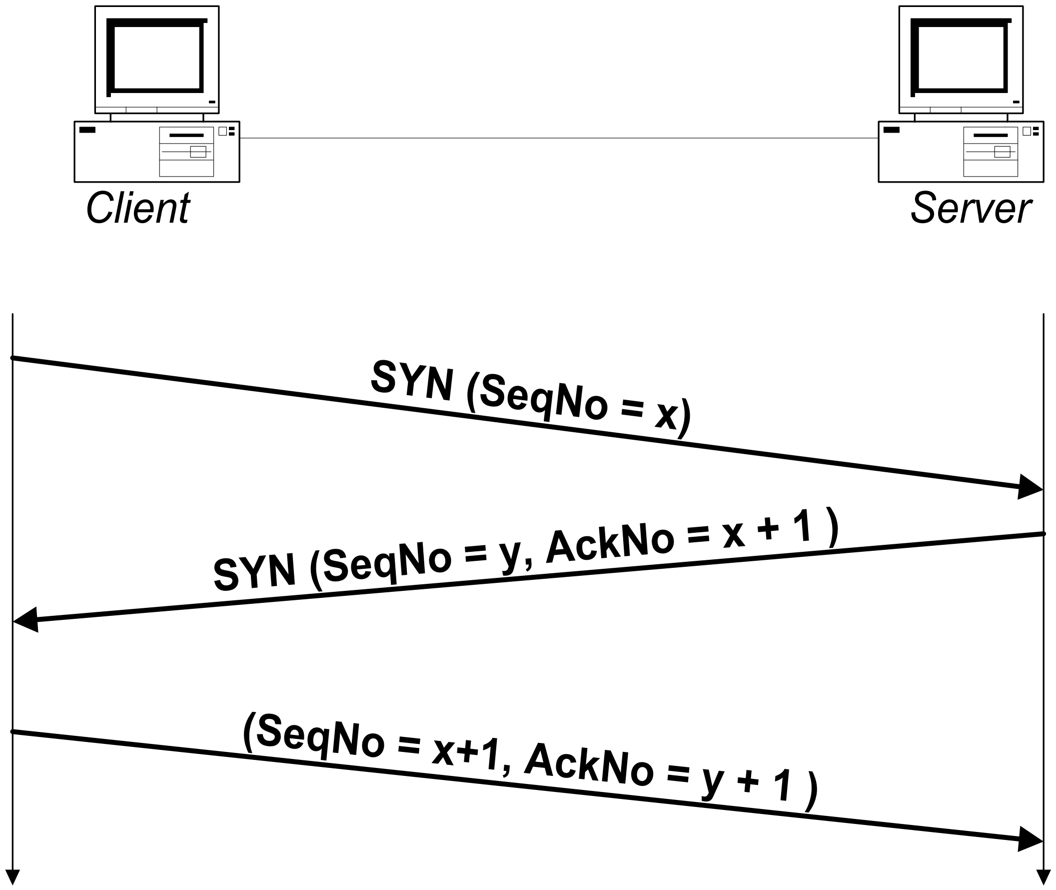
- TCP uses a three-way handshake to open a connection:
  1. **ACTIVE OPEN:** Client sends a segment with
     - SYN bit set *
     - port number of client
     - initial sequence number (ISN) of client
  2. **PASSIVE OPEN:** Server responds with a segment with
     - SYN bit set *
     - initial sequence number of server
     - ACK for ISN of client
  3. **Client acknowledges by sending a segment with:**
     - ACK ISN of server

  (* counts as one byte)

RFCs:
9293

Three-Way Handshake

RFCs:
9293



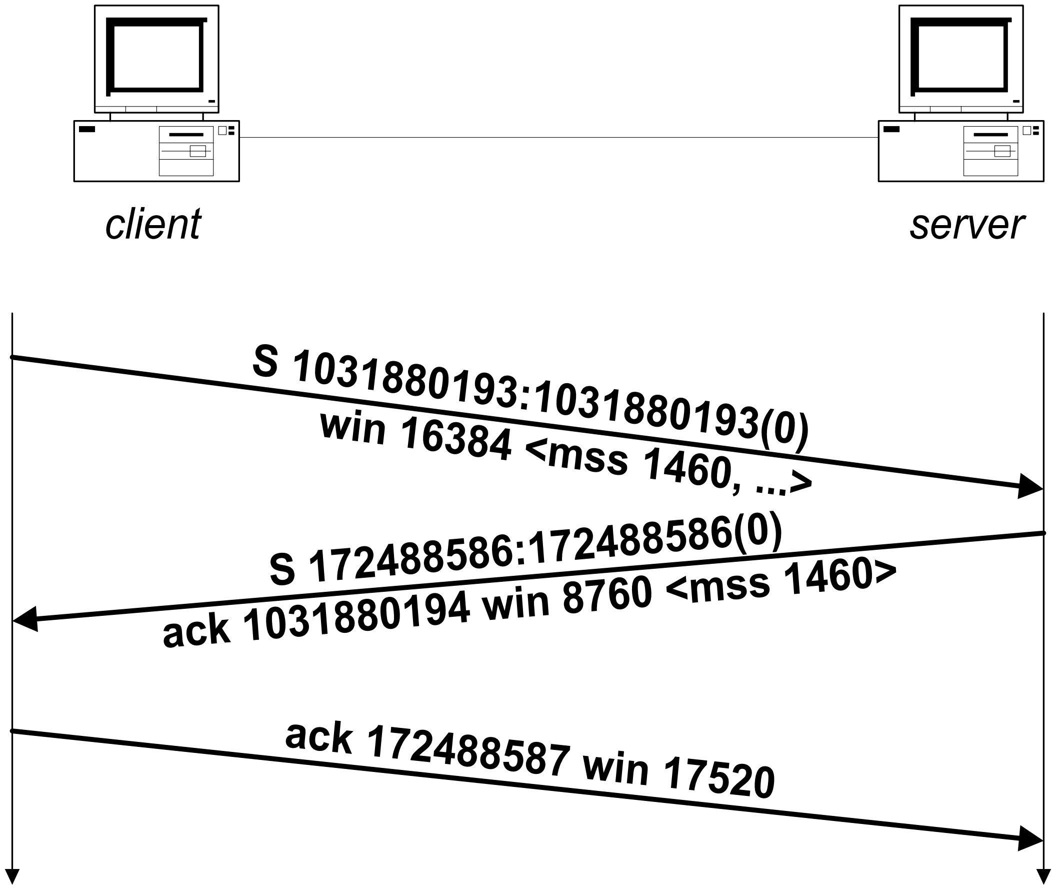
Three-Way Handshake *(continued)*

- 1st segment: client.1121 > server.23:
  - **Flags:** S
  - **SeqNo:** 1031880193:1031880193(0) win 16384
  - **Options:** <mss 1440,nop,wscale 0,nop,nop,timestamp>
- 2nd segment: server.23 > client.1121:
  - **Flags:** S, ACK
  - **SeqNo**: 172488586:172488586(0) AckNo: 1031880194 win 8760
  - **Options:** <mss 1440>
- 3rd segment: client.1121 > server.23:
  - **Flags:** ACK
  - **AckNo**: 172488587 win 17520
  - **Options:** .

RFCs:
9293

Three-Way Handshake *(continued)*

RFCs:
9293



TCP Connection Termination

RFCs:
9293

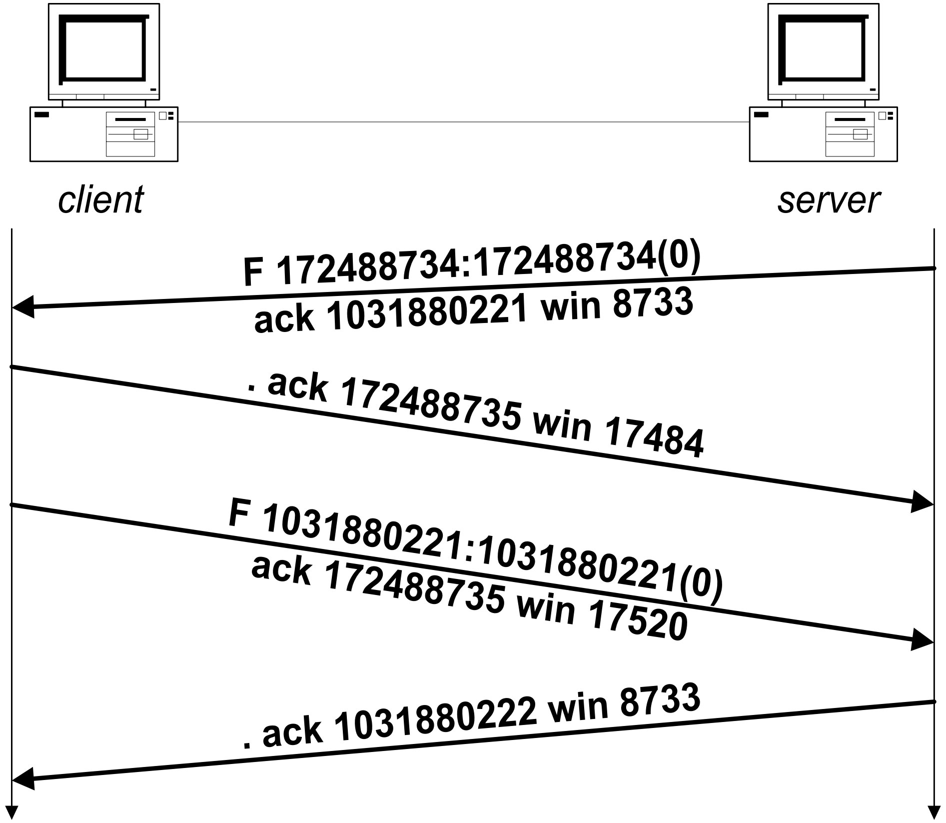
- Each end of the data flow must be shut down independently (“**half-close**”)
- If client or sender wish to terminate the connection they send a FIN segment
- The side that has sent the FIN segment cannot send new data
- First FIN segment can be sent by either client or server
- Each side of the connection must send a FIN segment to close the connection
- Four steps involved:
  - X sends a FIN to Y (active close)
  - Y ACKs the FIN,
    (*at this time:* Y can still send data to X, but X cannot send data to Y)
  - and Y sends a FIN to X (passive close)
  - X ACKs the FIN

TCP Connection Termination *(continued)*

RFCs:
9293

- 1^{ST} server.23 > client.1121:
  - **Flags:** FIN, ACK
  - **SeqNo:** 172488734 **AckNo:** 1031880221 win 8733
  - **Options:** .
- 2^{ND} segment: client.1121 > server.23:
  - **Flags:** ACK
  - **AckNo:** 172488735 win 17484
  - **Options:** .
- 3^{RD} segment: client.1121 > server.23:
  - **Flags:** FIN, ACK
  - **SeqNo:** 1031880221 **AckNo:** 172488735 win 17520
  - **Options:** .
- 4^{TH} segment: server.23 > client.1121:
  - **Flags:** ACK
  - **SeqNo:** 172488735 win 8733
  - **Options:** .

TCP Connection Termination *(continued)*



RFCs:
9293

TCP States

STATE	DESCRIPTION
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for Ack
SYN SENT	The client has started to open a connection
ESTABLISHED	Normal data transfer state
FIN WAIT 1	Client has said it is finished
FIN WAIT 2	Server has agreed to release
TIMED WAIT	Wait for pending packets (“2MSL wait state”)
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	Server has initiated a release
LAST ACK	Wait for pending packets

RFCs:
9293

TIME_WAIT State

RFCs:
9293

- When TCP does an active close, and sends the final ACK, the connection **must stay in the TIME_WAIT state for twice the maximum segment lifetime (2MSL)**
- The MSL is set to 2 minutes or 1 minute or 30 seconds
- By waiting in this state, the active closer is given a chance to resend the final ACK
  - Active closer will timeout after sending the FIN segment if no ACK is received
  - Then it will resend the FIN

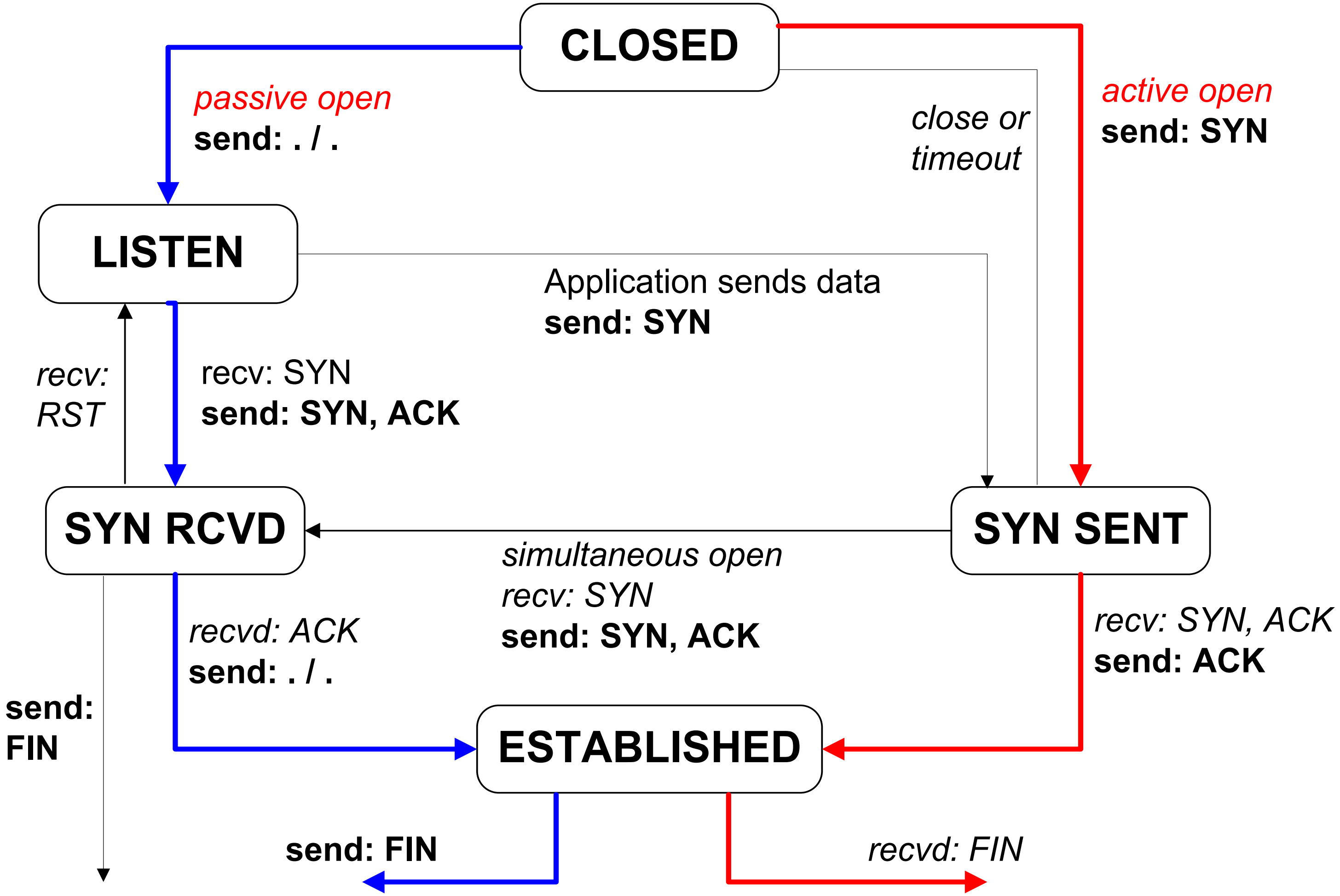
Resetting Connections

- Resetting connections is done by setting the RST flag in the TCP header
- **When is the RST flag set?**
  - Connection request arrives and no server process is waiting on the destination port
  - Abort (Terminate) a connection

    Causes the receiver to throw away buffered data. Receiver does not acknowledge the RST segment

RFCs:
9293

TCP: Opening A Connection



CLOSED

passive open
send: . / .

close or timeout

active open
send: SYN

LISTEN

Application sends data
send: SYN

recv: RST

recv: SYN
send: SYN, ACK

SYN RCVD

SYN SENT

simultaneous open
recv: SYN
send: SYN, ACK

recvd: ACK
send: . / .

recv: SYN, ACK
send: ACK

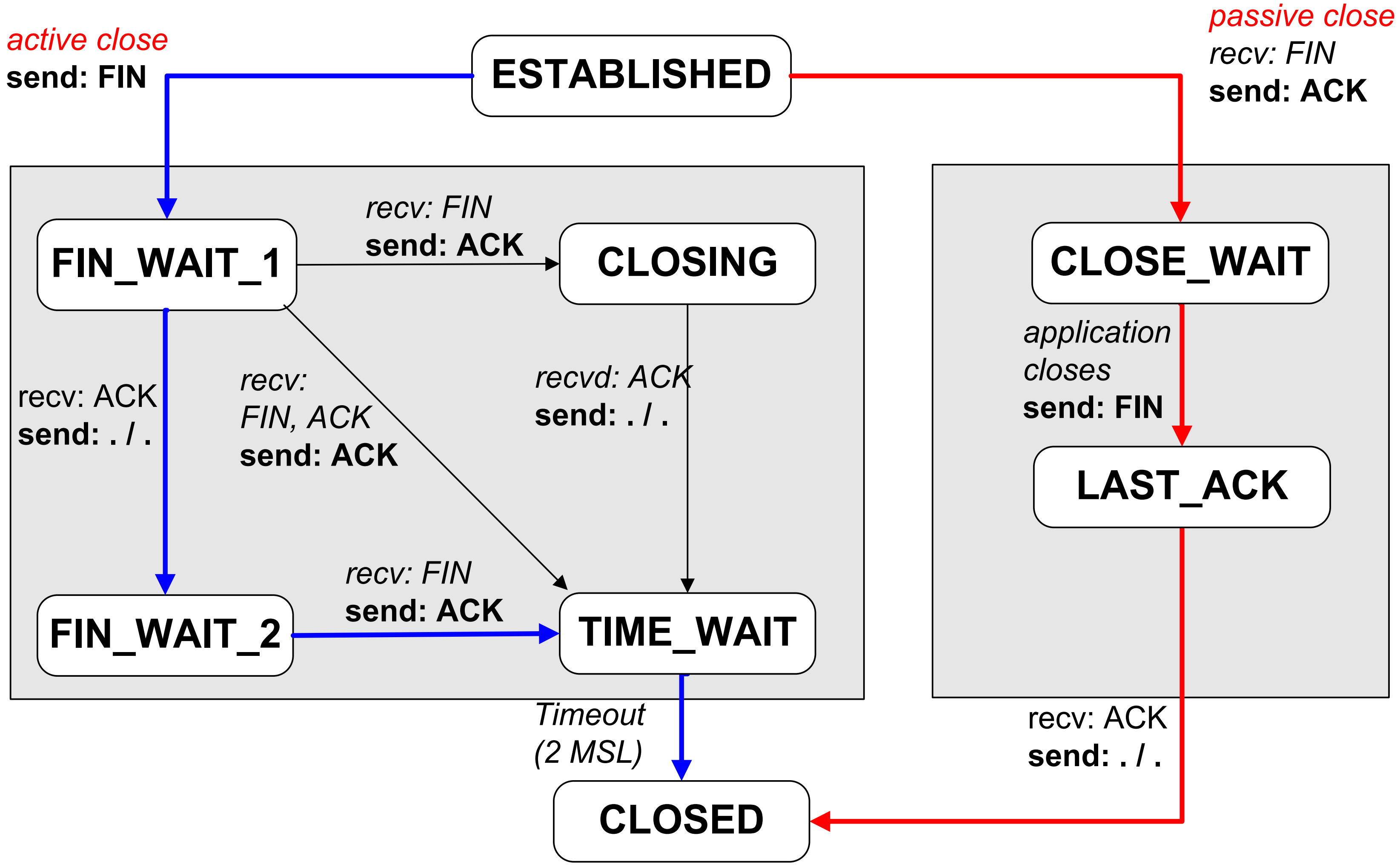
send: FIN

ESTABLISHED

send: FIN

recvd: FIN

TCP: Closing A Connection



active close
send: FIN

ESTABLISHED

passive close
recv: FIN
send: ACK

FIN_WAIT_1

recv: FIN
send: ACK

CLOSING

CLOSE_WAIT

recv: ACK
send: . / .

recv:
FIN, ACK
send: ACK

recvd: ACK
send: . / .

application
closes
send: FIN

LAST_ACK

recv: FIN
send: ACK

FIN_WAIT_2

TIME_WAIT

Timeout
(2 MSL)

recv: ACK
send: . / .

CLOSED

netstat

- Your new best friend
- Windows
  - -n: Show addresses and ports as numbers
  - -a: Show all connections and listening ports
  - -p: Specify a specific protocol
  - -o: Show owning PID
- Linux
  - -a: Show all connections and listening ports
  - -t: Show TCP
  - -u: Show UDP
  - -n: Don’t resolve names
  - -p: Show PID and process name

SS

- Linux has a new command line tool called ss to support much of the functionality that netstat had. The flags are generally similar to netstat
  - -t Show TCP
  - -u Show UDP
  - -x Show unix domain sockets
  - -a Show all sockets for given type. (All means both listening and non-listening
  - -p, --processes Show process associated with given socket
  - -n Don’t resolve service name

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

Lab 3

Transport Layer

ManTech

TASK 0: STOP!

- Set up Wireshark correctly!
- View ABSOULTE SEQ numbers, NOT relative!
- Edit -> Preferences - > Protocols -> TCP -> “Relative sequence numbers” = 0

TASK 1: TCP Reset

- With sniffer running…
  - Open a TCP connection between two linux hosts and let idle

```
peer1$ nc -l 8888
peer2$ nc <peer1addr> 8888
```

  - Select one of the endpoints as a dst
  - Using capture, determine the next seqno your dst expects from its peer

TASK 1: TCP Reset *(continued)*

- Construct Ethernet header, IP header, and TCP header with that seqno and the RST bit set
- Using helper function, send to dst

```
>>> cno.rawsend_cksum_ipv4(etherhdr + iphdr + tcphdr, dev)
```

- Attempt to send data across connection
- What happens?
- Team up with the people around you, reset their connections

TASK 1: TCP Reset *(BONUS)*

- Make your script be able to send the reset packet automatically.
- You can configure it with the src and dest but it should be able to determine the proper sequence number on it's own

Lesson Review

- Questions
- Review
- Summary

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

Sockets

Objectives

Given a workstation, device, and/or technical documentation, the student will be able to:

- Learning Objective
  - Use Sockets to create simple network tools.
- Enabling Objective
  - Identify different types of sockets
  - Use UDP and TCP sockets to create network tools

What is a Socket?

- A Sockets implementation provides an API for user-mode applications to use kernel-mode network services (Data-Link, Network, Transport layers)
- A socket is a descriptor/handle on which an application performs socket operations

Types of Internet Sockets

- Datagram Sockets
  - Connectionless
  - Uses UDP to send packets to the remote host with no guarantee of order of delivery or even any delivery at all
- Stream Sockets
  - Connection-Oriented
  - Uses TCP for reliable delivery of packets in a sequenced order
- Raw Sockets
  - Because the packet header is delivered with the packet (unlike in Stream and Datagram sockets), users can access these packet headers, which can lead to security issues such as IP Address Spoofing and Denial-of-Service attacks

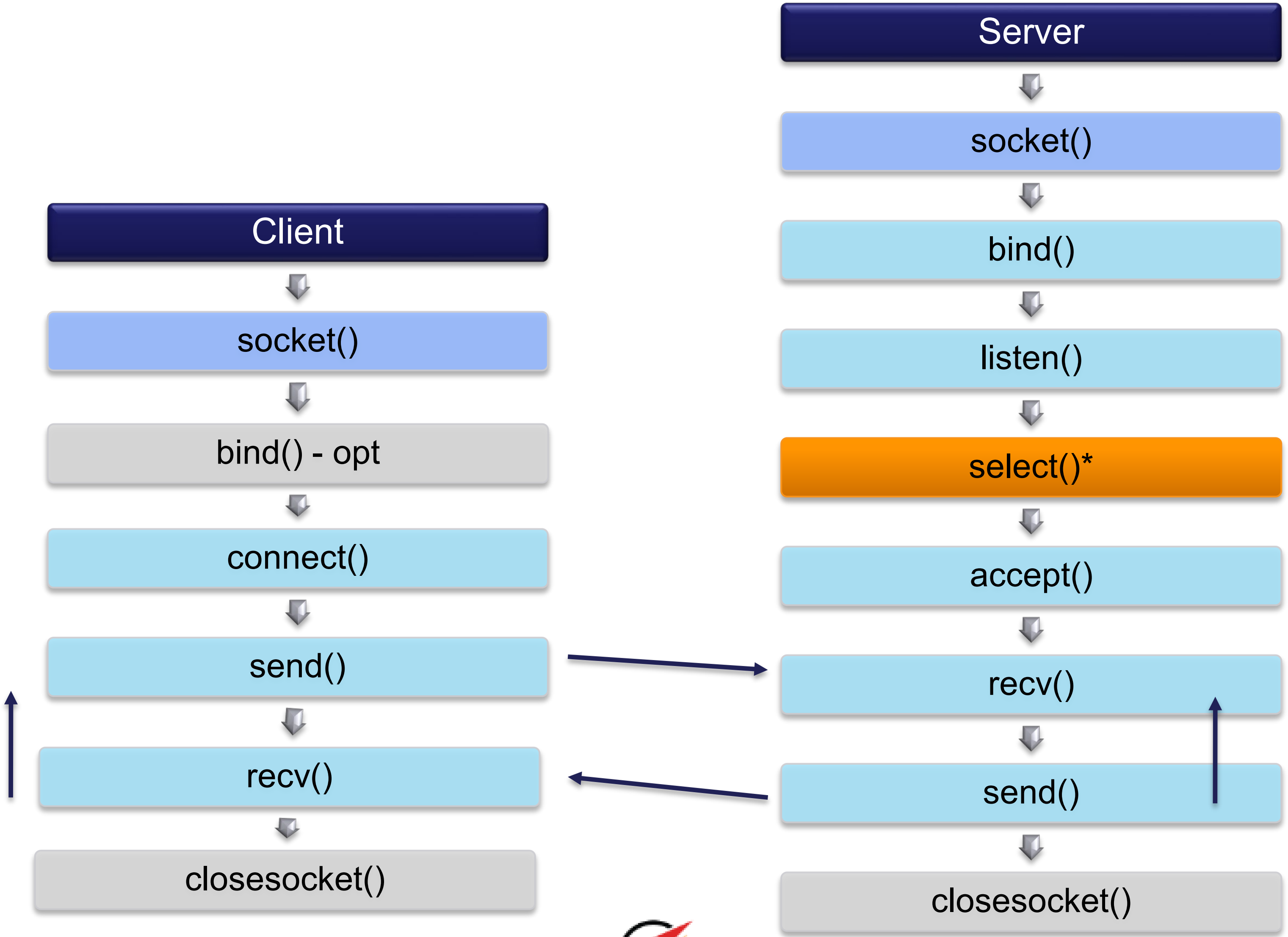
Most Important Functions

FUNCTION	DESCRIPTION
accept	Accept connection from kernel accept queue
bind	Use specific local address/port
connect	Connect to some address/port
getsockopt	Read kernel-level socket options
htonl	Convert 32bit value from host to network byte order
htons	Convert 16bit value from host to network byte order
listen	Go to listening state with given accept queue len
ntohl	Convert 32bit value from network to host byte order
ntohs	Convert 16bit value from network to host byte order

Most Important Functions *(continued)*

FUNCTION	DESCRIPTION
recv	Dequeue stream bytes from kernel rcvbuf to user buf
recvfrom	Dequeue datagram from kernel rcvbuf to user buf
send	Queue stream bytes from user buf to kernel sndbuf
sendto	Queue datagram from user buf to kernel sndbuf
setsockopt	Write kernel-level socket options
shutdown	Shutdown either or both side of connection
socket	Create socket
If_nametoindex	Return a network interface index number corresponding to an interface name
Inet_pton	Convert and ip address from a it’s family specific string format to a packed binary format

TCP Client & Server



Server

socket()

bind()

listen()

select()*

accept()

recv()

send()

closesocket()

Client

socket()

bind() - opt

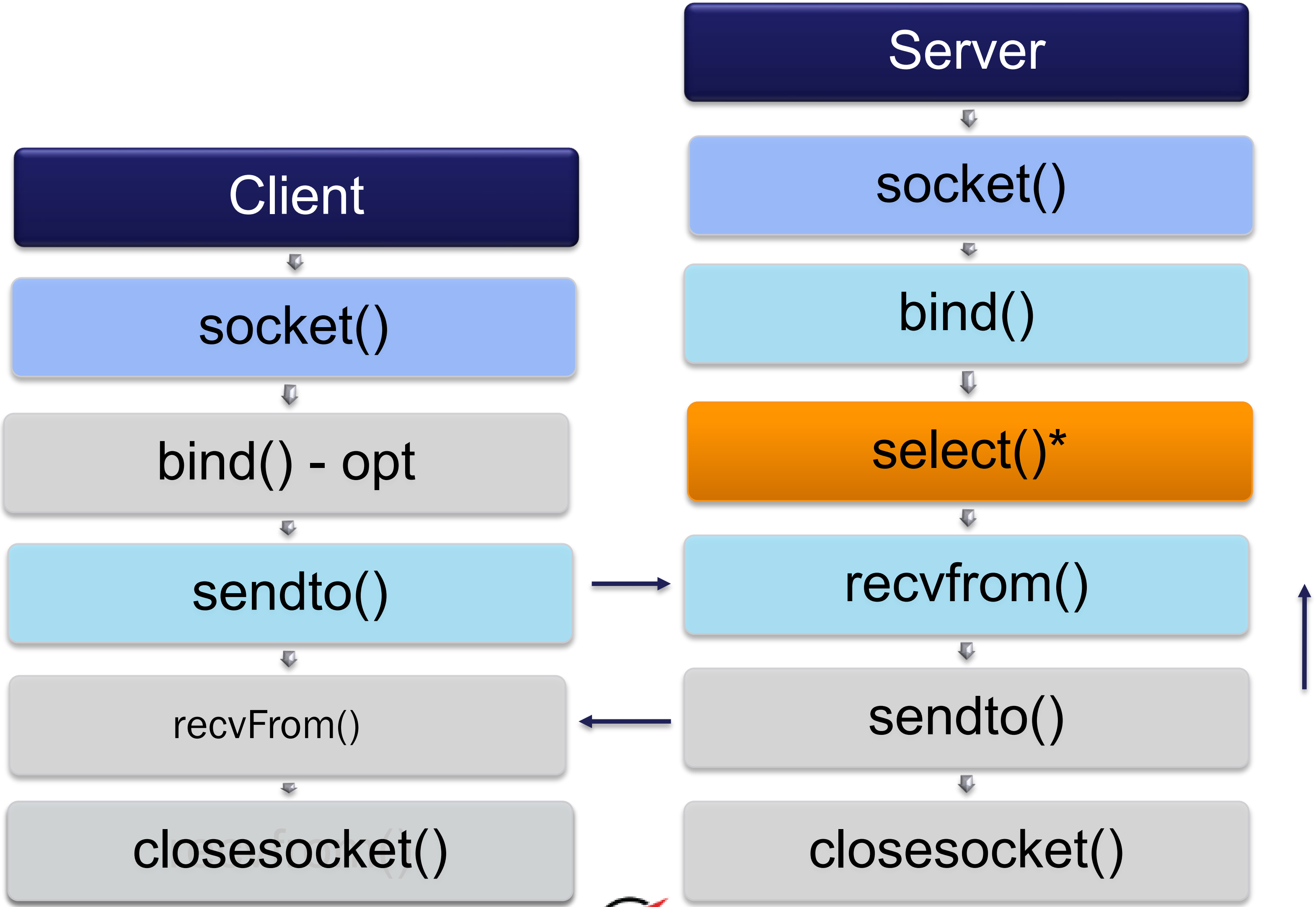
connect()

send()

recv()

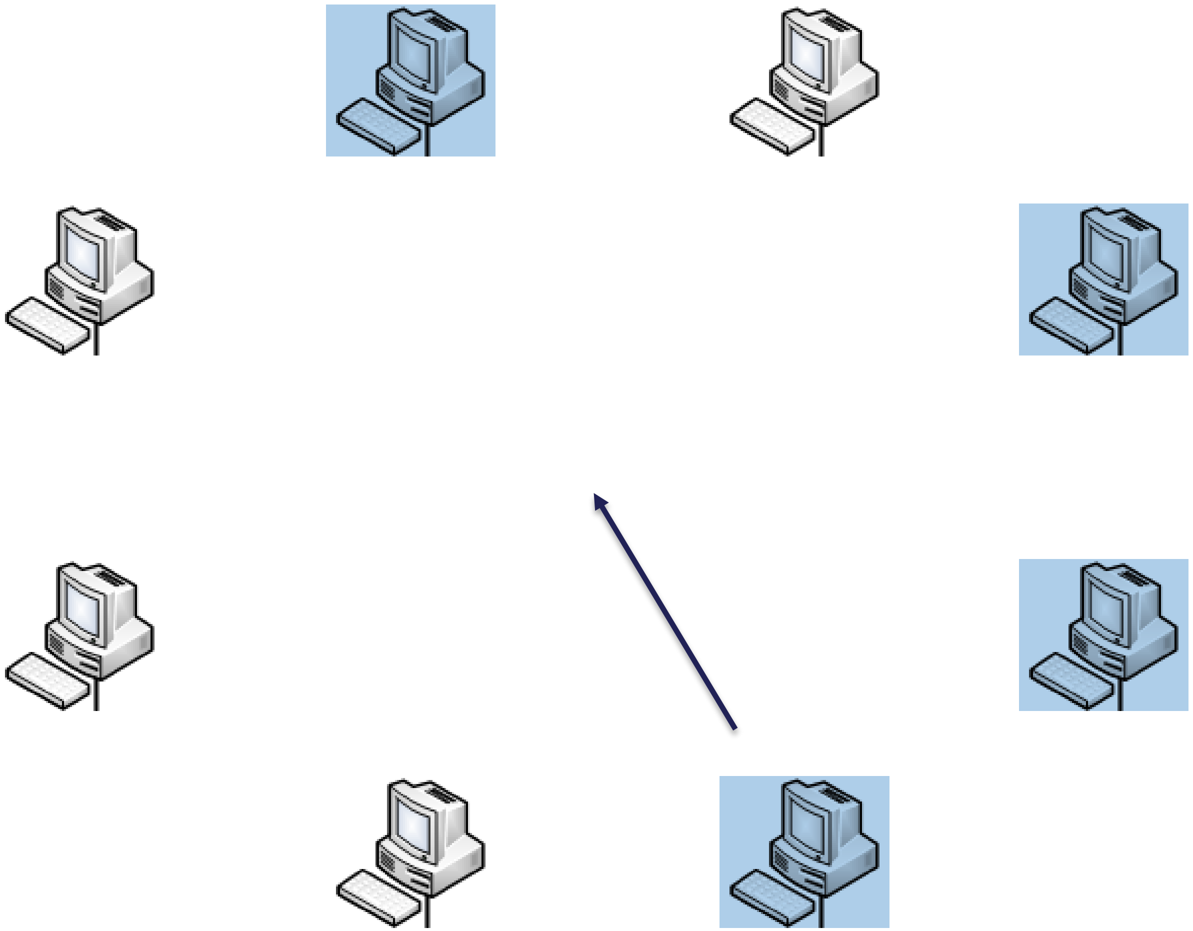
closesocket()

UDP Client & Server



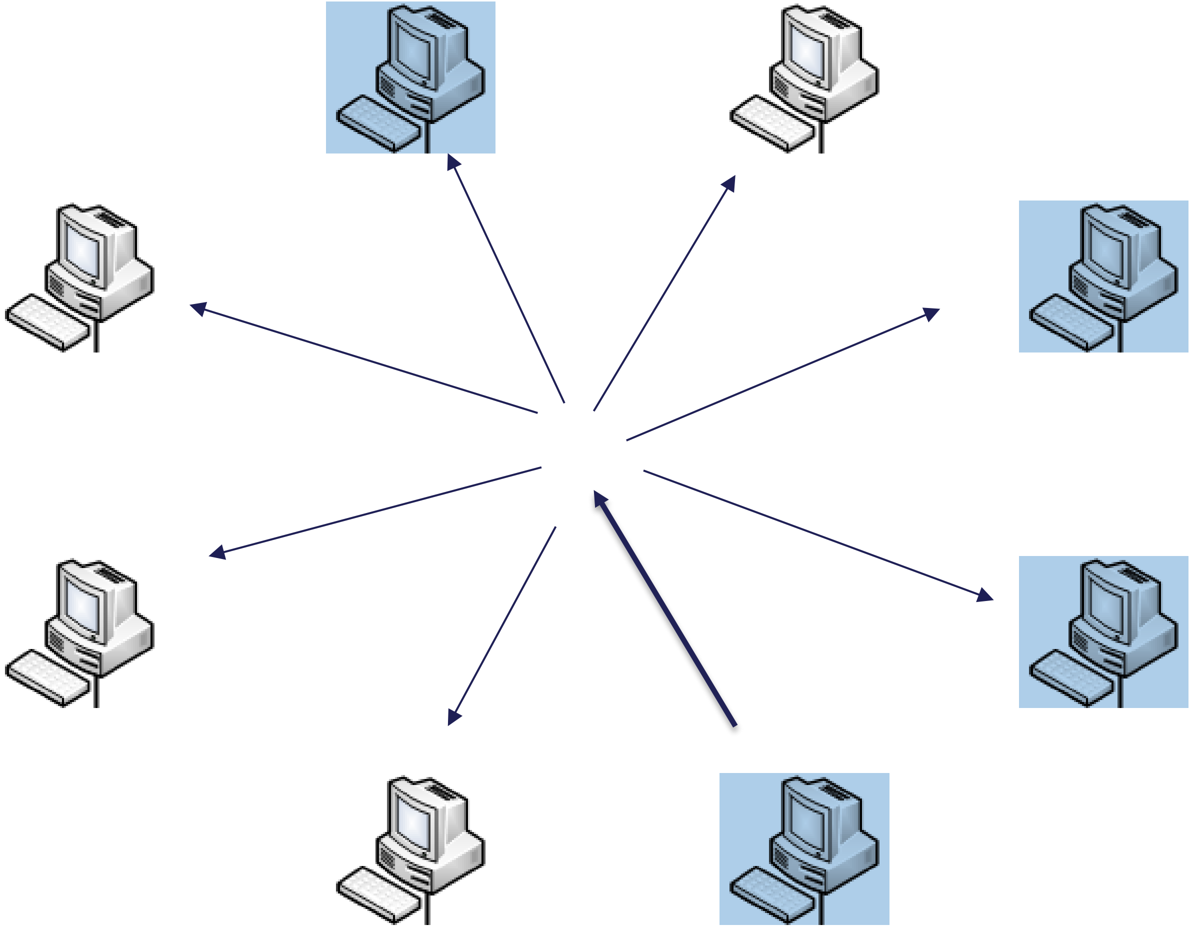
Multicast

ff02::1:12345



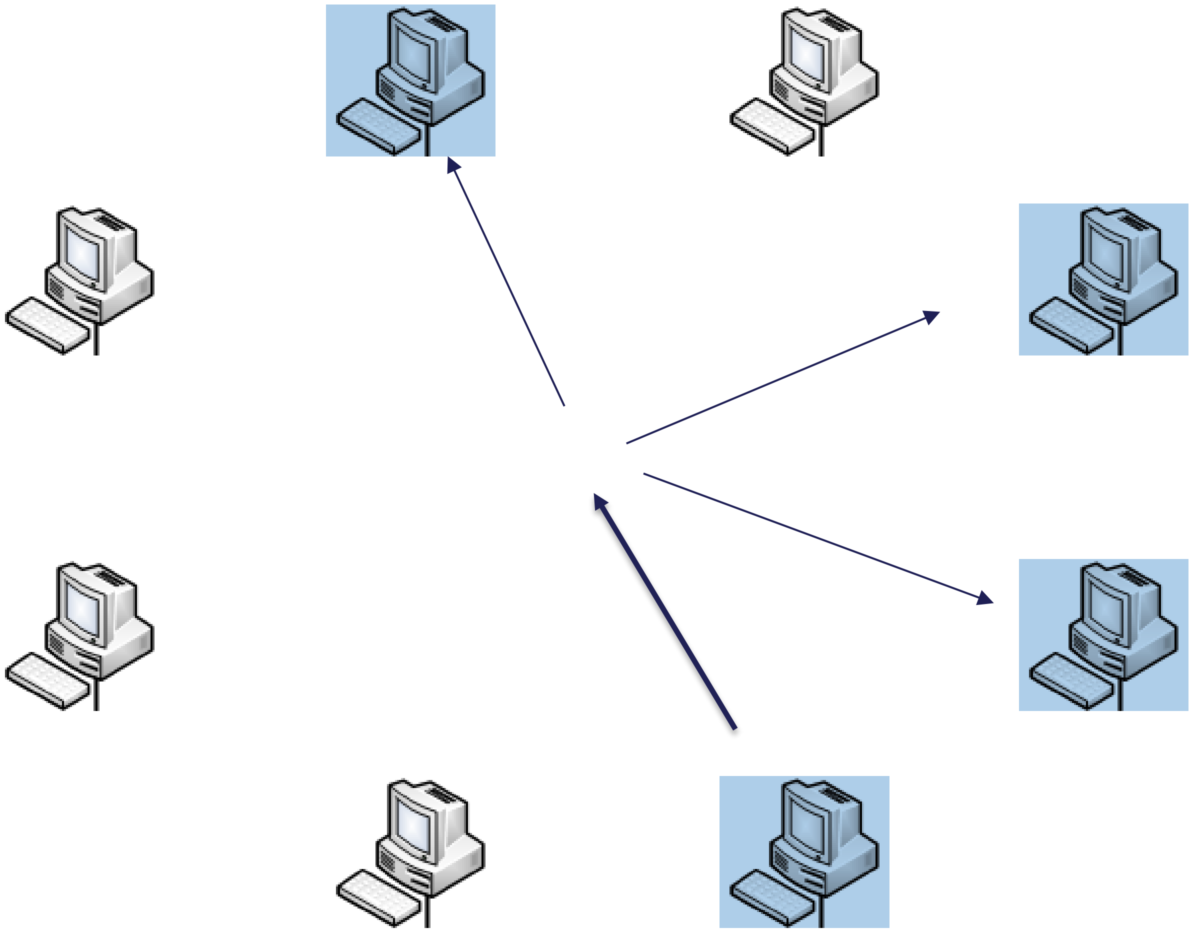
Multicast

ff02::1:12345



Multicast

ff02::1:12345



Multicast

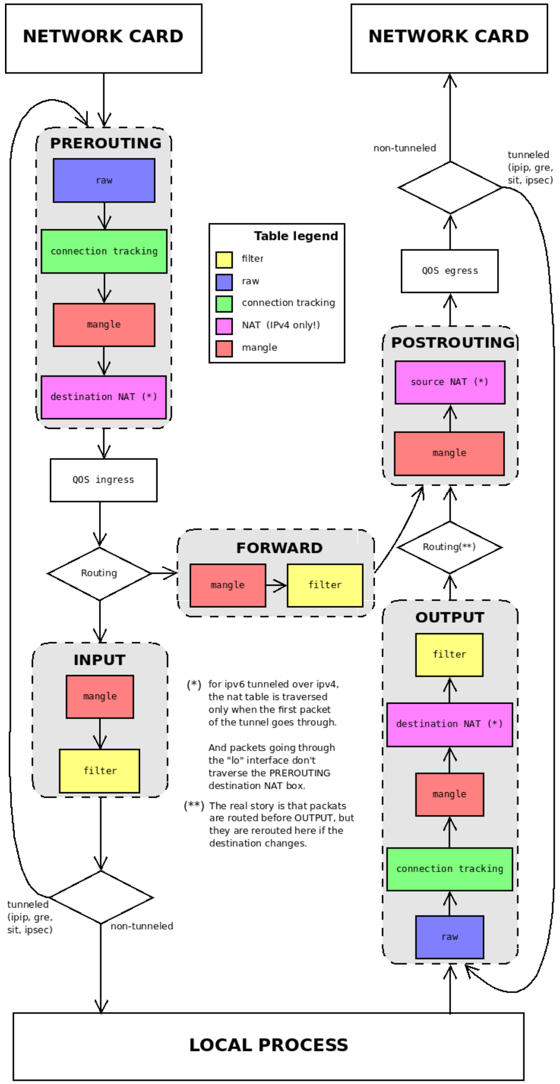
- Can a computer not in the multicast group send a packet to the multicast address? Why or why not?
- Can Wireshark see multicast packets? Why or why not?

iptables

- iptables for Linux
  - ip6tables (IPv6)
  - iptables (IPv4)
  - arptables (ARP)
  - ebtables (Ethernet frames)
- Gentoo
  - /var/lib/iptables/rules-save
  - /etc/init.d/iptables [stop|start|restart]
- Kernel 3.13 – Introduced “nftables”, a replacement
- Kernel 3.15 – Expected to be fully implemented

iptables

- Tables (5, default is 'filter')
- Chains (3)
    - Default policy
- Rules
- View this picture from the network share, it'll be easier to see →



(*) for ipv6 tunneled over ipv4, the nat table is traversed only when the first packet of the tunnel goes through.

And packets going through the "lo" interface don't traverse the PREROUTING destination NAT box.

(**) The real story is that packats are routed before OUTPUT, but they are rerouted here if the destination changes.

iptables – default chains

- **“PREROUTING”**: Packets will enter this chain before a routing decision is made
- **“INPUT”**: Packet is going to be locally delivered. Has nothing to do with a listening process yet. `ip route show table local`
- **“FORWARD”**: All packets that have been routed and were not for local delivery will traverse this chain
- **“OUTPUT”**: Packets sent from the machine itself will be visiting this chain
- **“POSTROUTING”**: Routing decision has been made. Packets enter this chain just before handing them off to the hardware

iptables - tables

- **Filter** – This is the table most people use when using iptables. It makes decisions on whether the packet should continue or not
- **NAT** – Used to implement network address translation rules
- **Mangle** – Used to modify a packet in some way (i.e. update a TTL header etc)
- **Raw** – Used to allow packets to opt out of connection tracking
- **Security** – Used to set SELinux marks on packets

iptables – Useful commands

- List iptables:
  - iptables –S
  - iptables –L [INPUT,FORWARD,OUTPUT]
  - iptables –L --line-numbers
- Delete rule:
  - iptables –D <chain> <line number> (deletes a rules by line number)
  - iptables –D <chain> <rule> (deletes a rule based on the rule)
  - iptables –F <chain> (deletes all rules in the chain)
- Set default policy:
  - iptables –P <chain> [ACCEPT,DROP,REJECT]

iptables – Useful commands

- Add a rule:
  - iptables -A <chain> <lots of other stuff>
  - iptables -A INPUT -s 145.145.231.5 -j DROP
    - Drop packets from 145.145.231.5
  - iptables -A OUTPUT -p udp --dport 53 -j REJECT
    - Reject udp packets that are being sent from this computer going to port 53
  - iptables -A INPUT -p tcp -s 15.15.15.0/24 --dport 22 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
    - Accept new or established connections that are tcp from the subnet 15.15.15.0/24 and are destined for port 22
  - iptables -A OUTPUT -p tcp --sport 873 -m conntrack --ctstate ESTABLISHED -j ACCEPT
    - Allow established tcp packets that this computer is sending out from source port 873
  - iptables -A INPUT -p udp -j REJECT --reject-with icmp-port-unreachable
    - reject all udp packets coming to INPUT with an icmp port unreachable message

LINUX
CNO
Programming

ADVANCED CYBER TRAINING PROGRAM

Lab 4

Sockets

TASK 1: iptables

- mkdir /tmp/public
- cd /tmp/public
- Start a webserver

```
python2 -m SimpleHTTPServer [port]
python3 -m http.server [port]
```

- Lab: configure iptables to only allow your neighbor to access (by IP address) (Hint: man iptables)

```
# echo ‘my secret’ > file.txt
```

  - Only your neighbor should be able to browse to your web server and see your secret!

TASK 1: TCP Client

- Start a TCP listener using nc

```
$ nc -l 8888
```

- In a Python interpreter…
  - Open a TCP socket

```
>>> import socket
>>> s = socket.socket(socket.AF_INET,
    socket.SOCK_STREAM)
```

- Connect to your listener

```
>>> s.connect((“<ip>”, 8888))
```

- Send data to listener

```
>>> s.send(b“hello, my name is packet”)
```

TASK 2: UDP Client

- Start a UDP listener using nc

```
$ nc -u -l 8888
```

- Open a UDP socket

```
>>> import socket
>>> s = socket.socket(socket.AF_INET,
    socket.SOCK_DGRAM)
```

- Connect to your listener
  - Yes udp does accept connect. It will just set the default address to use when calling send and recv

```
>>> s.connect((“<ip>”, 8888))
```

- Send data to listener

```
>>> s.send(b“hello, my name is packet”)
```

- Alternatively, use s.sendto()

```
>>> s.sendto(b“foobar”, (“<ip>”, 8888))
```

TASK 3: TCP/UDP Client/Server

- Use Python's socket interface to implement a TCP server that can receive messages from your TCP client. The TCP server should display messages received
- Do the same with UDP
- Now that you have a client and server do a half-shutdown client socket with your server socket, and view in Wireshark

```
>>> s.shutdown(socket.SHUT_WR)
```

- **Bonus** – Allow the TCP server to service multiple TCP clients

TASK 4: Multicast Client/Server

- Write a program that can send and receive to/from an instructor-assigned multicast address
    - ff02::<student>: <student>:<student>:
- There will be sample/starter code on the network share

LINUX
CNO
Programming

ADVANCED CYBER TRAINING PROGRAM

Port Scanning

Objectives

Given a workstation, device, and/or technical documentation, the student will be able to:

- Learning Objective
  - Understand how TCP and UDP port scans are used to discover the state of a TCP/UDP port

- Enabling Objective
  - Use the python sockets API to write a port scanner
  - Use the python raw sockets API to write a port scanner

Types of Port Scans

- Connect Scan (SYN, SYN/ACK, ACK)
- SYN Scan (SYN) aka half-open scanning
- NULL Scan (No flags)
- FIN Scan (FIN flag)
- Christmas Tree Scan (URG PSH FIN)
- ACK Scan (determines firewall rules)
- http://nmap.org/book/man-port-scanning-techniques.html

Connect() Scan

- Open?
  - SYN->
  - <-SYN/ACK
  - ACK->
    - Server.Log("Connection established with $IP"); - NOISY!
- Closed?
  - SYN->
  - <-RST
- Filtered?
  - SYN->
  - <nothing returned> OR ICMP Port Unreachable

SYN Scan / Half-Open Scan

- Open?
  - SYN->
  - <-SYN/ACK
    - Server will not log failed 3 way handshakes
    - Requires root, as opposed to Connect() scans
- Closed?
  - SYN->
  - <-RST
- Filtered?
  - SYN->
  - <nothing returned> OR ICMP Port Unreachable

NULL, FIN, and Xmas Scans

- Helps differentiate between open and closed ports
- “if the [destination] port state is CLOSED…an incoming segment not containing a RST causes a RST to be sent in response.” Then the next page discusses packets sent to open ports without the SYN, RST, or ACK bits set, stating that: “you are unlikely to get here, but if you do, drop the segment, and return.” - Page 65 of RFC 9293 (TCP)
- Any packet sent not containing a SYN, ACK, or RST, answered with a RST = closed
- No answer = open|filtered
- Answered with an ‘ICMP unreachable error ‘ = filtered
- Target MUST be fully RFC 9293 compliant - They may not be

ACK Scan

- Does NOT determine open or open|filtered ports
- Used to map out firewall rules
- Send an ACK flag only
- Unfiltered systems will reply with a RST
  - Note, this is targeting the ESTABLISHED,RELATED rule, we know nothing about if it is open or closed
  - Labeled as ‘unfiltered’, as the ACK obviously got to the system
- No response, or ICMP responses, means the port is filtered

UDP Scan

- Send a UDP packet and check for ICMP port unreachable error
- If it is a different ICMP error message than the port is filtered
- OS's will rate limit ICMP error message so that makes scanning slow
- If scanning a port with a well known protocol try sending a packet with appropriate data

LINUX
CNO
Programming

Lab 5

Port Scanners

TASK 1: Socket Based Port Scanner

- Use the python sockets API to write a TCP connect port scanner.
- IP(s) and ports to scan should be given on the command line or via a text file(s)
- Account for both IPv4 addresses and IPv6 addresses

TASK 2: Raw Socket Based Port Scanner

- Use the python raw sockets API to write a TCP port scanner
  - IP(s) and ports to scan should be given on the commandline or via a text file(s)
  - Account for both IPv4 addresses and IPv6 addresses
  - You can use python raw sockets API to make a listener

```
s = socket.socket( socket.AF_PACKET , socket.SOCK_RAW , socket.ntohs(0x0003))

S.bind((‘ens33’, 0))
```

  - How fast can you make it?
    - Check out cProfile in python
- Note:
  - Enable checksum verification in wireshark

Bonus: Write a UDP port scanner

- Use the python raw sockets api to write a UDP port scanner
- Same idea as the previous task

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

Application Layer

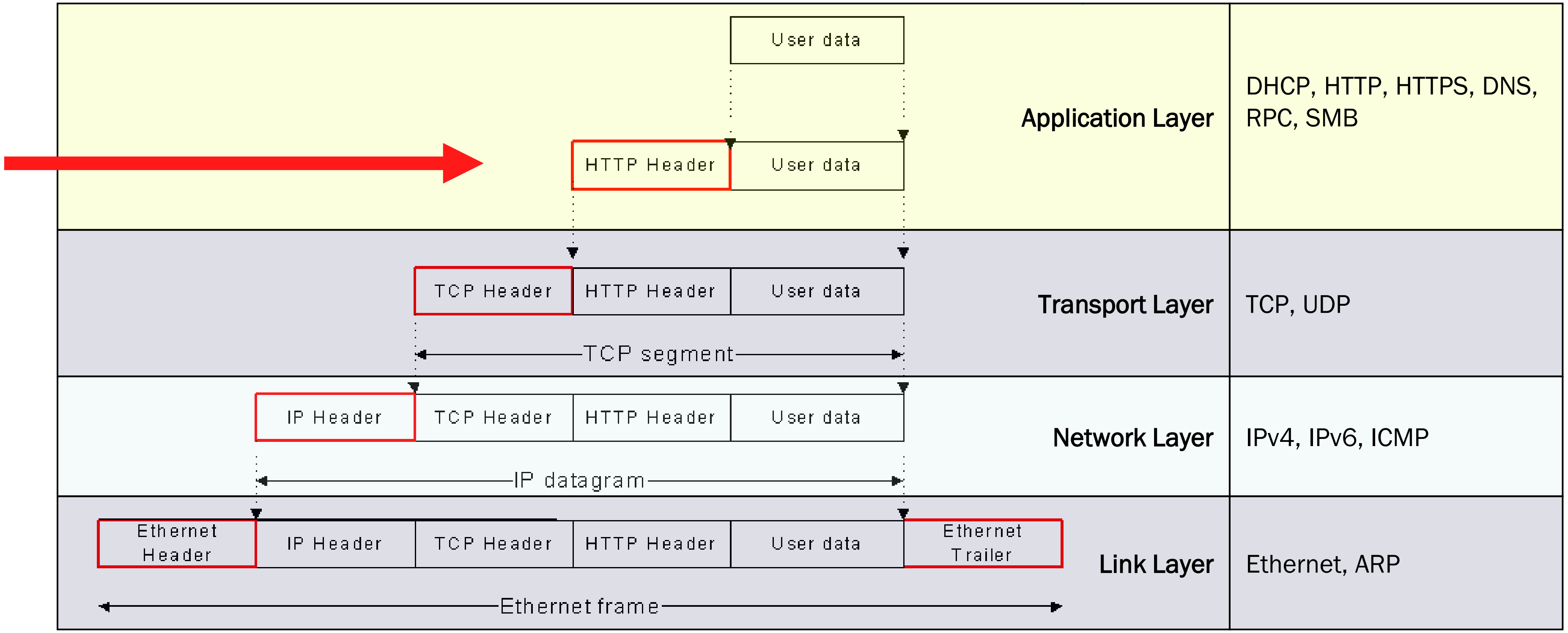
Objectives

Given a workstation, device, and/or technical documentation, the student will be able to:

- Learning Objective
  - Use common Application Layer protocol specifications to analyze, interpret, and construct network traffic
- Enabling Objectives
  - Describe how DHCP is used to obtain configuration information for operation in an IP network
  - Describe how HTTP provides communication between clients and servers
  - Describe how HTTPS provides secure communication between clients and servers
  - Create HTTP requests and analyze the responses
  - Describe DNS design and function from a network perspective
  - Use existing tools to create DNS requests and interpret the responses

Application Layer

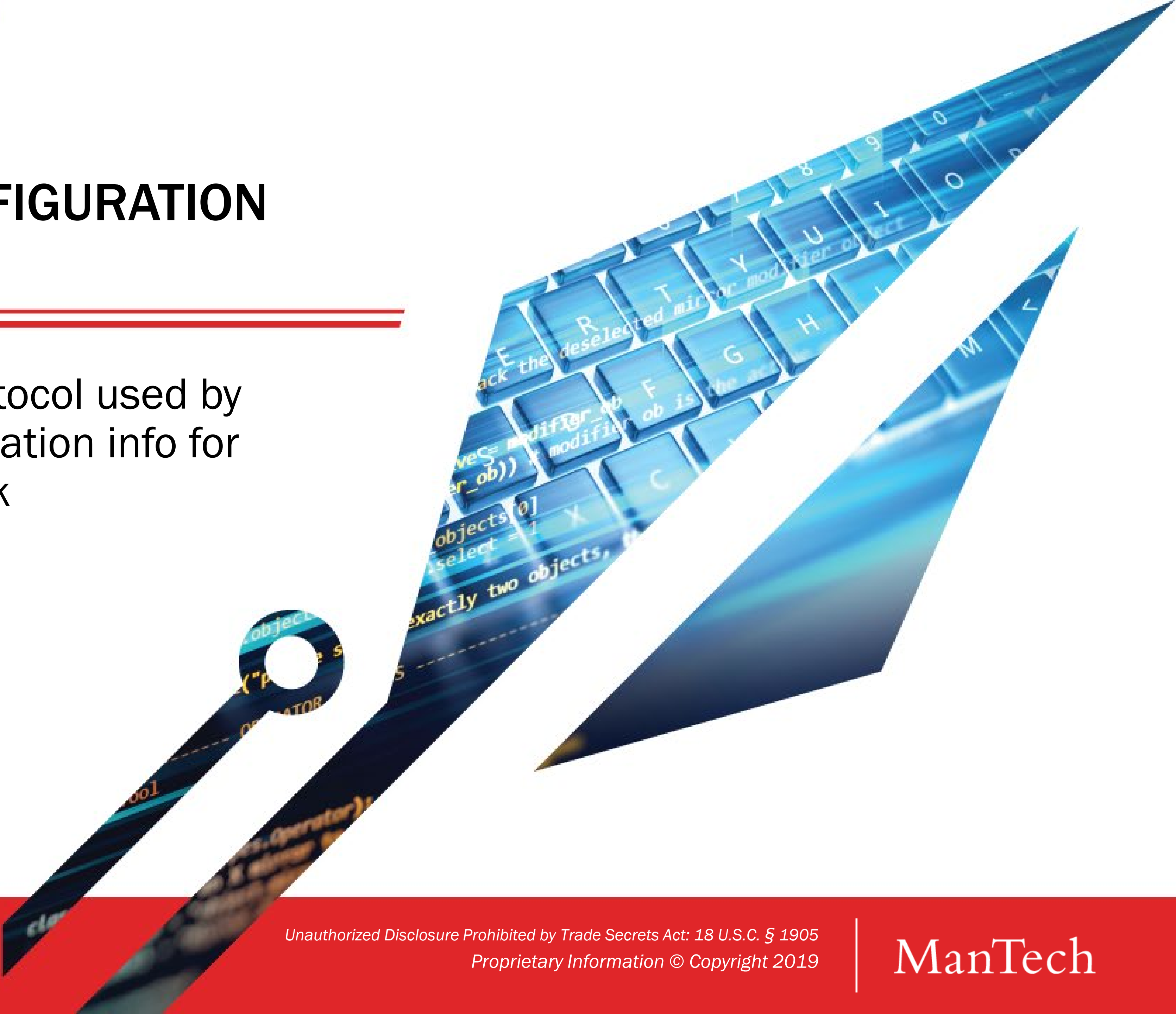
- Contains all protocols and methods that fall into the realm of process-to-process communication



Application Layer	DHCP, HTTP, HTTPS, DNS, RPC, SMB
Transport Layer	TCP, UDP
Network Layer	IPv4, IPv6, ICMP
Link Layer	Ethernet, ARP

DYNAMIC HOST CONFIGURATION PROTOCOL (DHCP)

- A network application protocol used by devices to obtain configuration info for operation in an IP network



Dynamic Allocation of IP Addresses

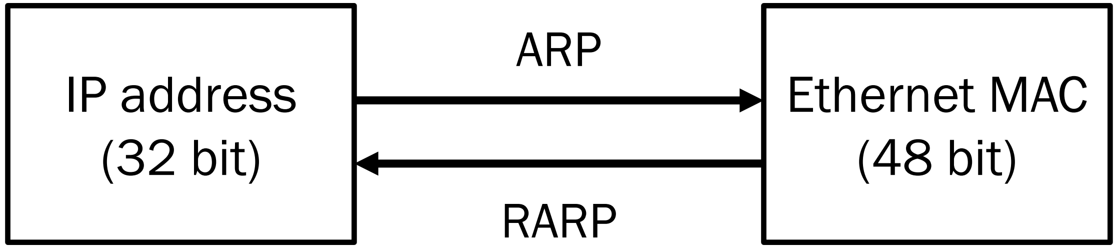
- Dynamic assignment of IP addresses is desirable for several reasons:
    - IP addresses are assigned on-demand
    - Avoid manual IP configuration
    - Support mobility of laptops
- Four Protocols:
    - RARP (until 1985, no longer used)
    - BOOTP (1985-1993)
    - DHCP (since 1993)
    - DHCPv6 (since 2003)
- Only DHCP is widely used today

RFCs:
903, 1931, 2113, 8415

Reverse Address Resolution Protocol (RARP)

- RARP is no longer used
- Works similar to ARP
- Broadcast a request for the IP address associated with a given MAC address
- RARP server responds with an IP address
- Only assigns IP address (not the default router and subnetmask)

RFCs:
903, 1931

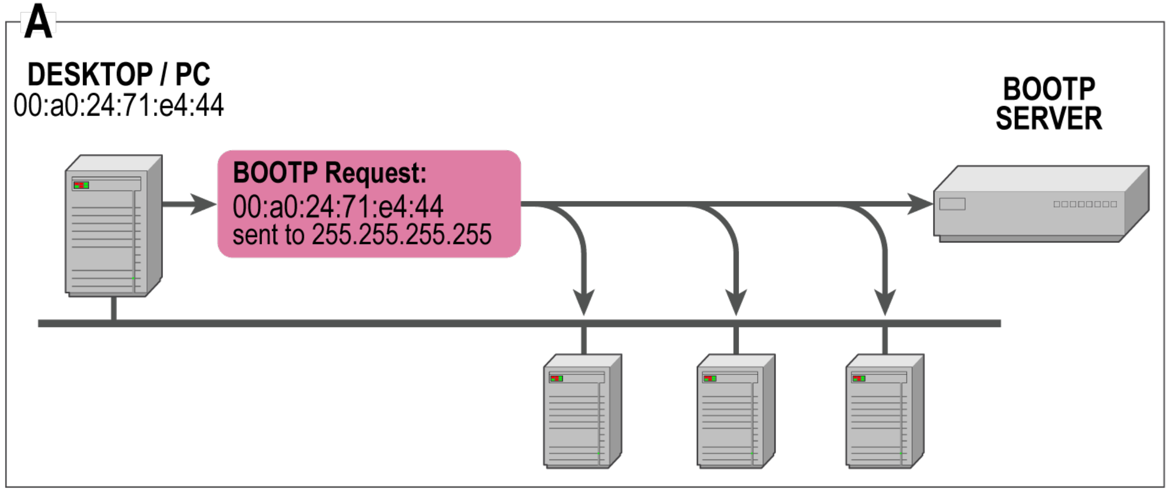


BOOTstrap Protocol (BOOTP)

RFCs:
951, 2132, 1534, 1497

- Host can configure its IP parameters at boot time
- 3 services:
  - IP address assignment
  - Detection of the IP address for a serving machine
  - The name of a file to be loaded and executed by the client machine (boot file name)
- Not only assigns IP address, but also default router, network mask, etc.
- Sent as UDP messages (UDP Port 67 (server) and 68 (host))
- Used limited IPv4 broadcast address (255.255.255.255):
  - These addresses are never forwarded

BOOTP Interaction

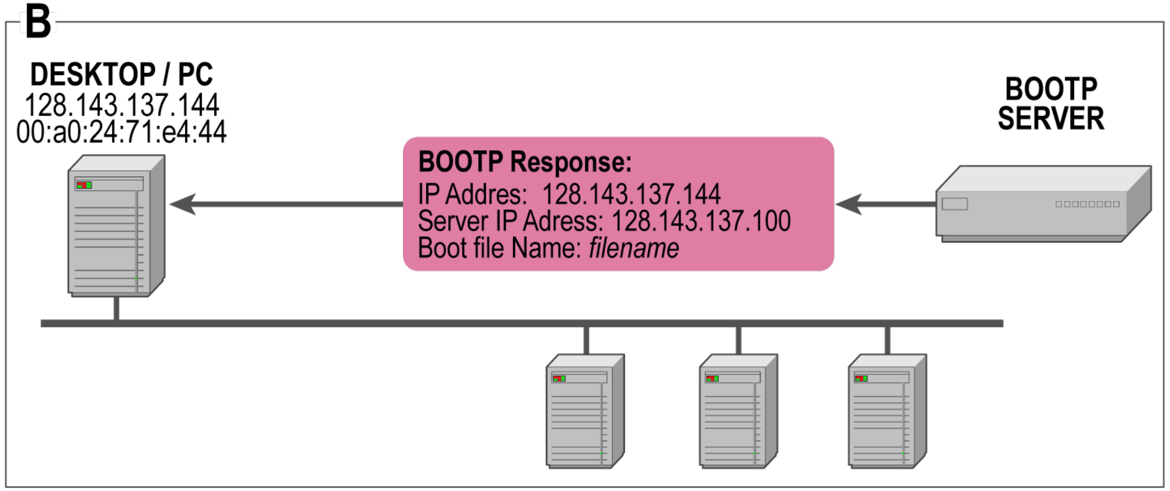


A

DESKTOP / PC
00:a0:24:71:e4:44

BOOTP SERVER

BOOTP Request:
00:a0:24:71:e4:44
sent to 255.255.255.255

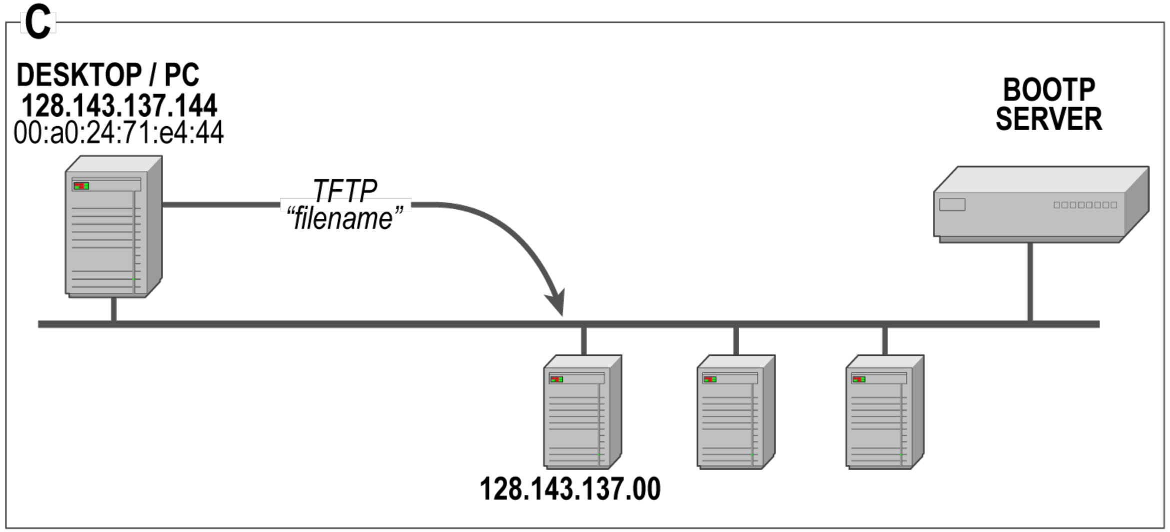


B

DESKTOP / PC
128.143.137.144
00:a0:24:71:e4:44

BOOTP SERVER

BOOTP Response:
IP Addres: 128.143.137.144
Server IP Adress: 128.143.137.100
Boot file Name: *filename*



C

DESKTOP / PC
128.143.137.144
00:a0:24:71:e4:44

BOOTP SERVER

TFTP
"filename"

128.143.137.00

- BOOTP can be used for downloading memory image for diskless workstations
- Assignment of IPv4 addresses to hosts is static

Dynamic Host Configuration Protocol (DHCP)

RFCs:
2131, 2132, 5107

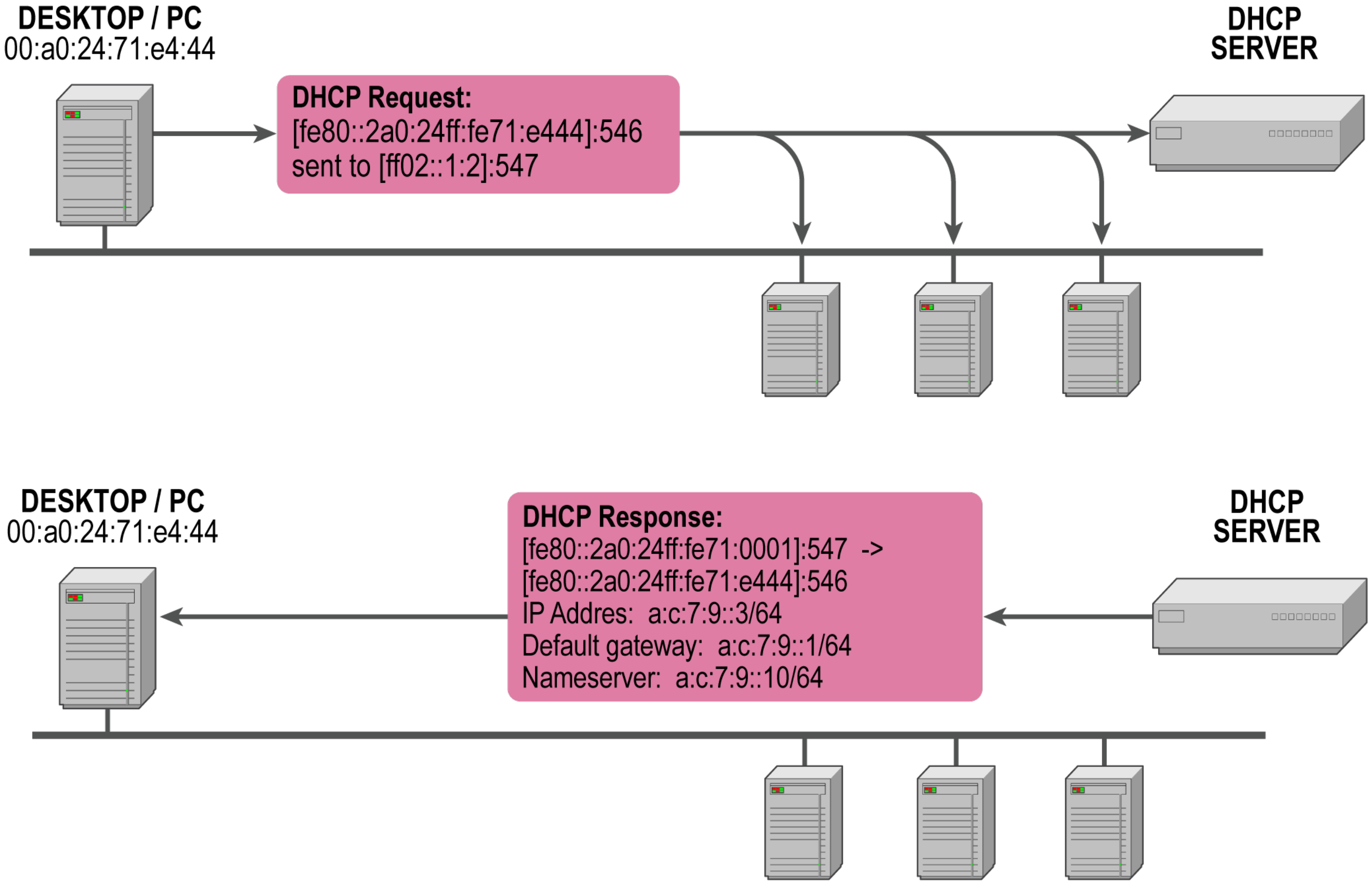
- Designed in 1993
- An extension of BOOTP (Many similarities to BOOTP)
- Same port numbers as BOOTP
- Extensions:
  - Supports temporary allocation (“leases”) of IP addresses
  - DHCP client can acquire all IP configuration parameters
- DHCP is the preferred mechanism for dynamic assignment of IP addresses
- DHCP can interoperate with BOOTP clients

Dynamic Host Config. Protocol v6 (DHCPv6)

- Designed in 2003
- An extension to the DHCP protocol to enable stateful addressing of IPv6 addresses
- Uses following multicast addresses:
  - All_DHCP_Relay_Agents_and_Servers (FF02::1:2) – link scoped
  - All_DHCP_Servers (FF05::1:3) – site-scoped
- Clients listen on UDP 546
- Servers listen on UDP 547

RFCs:
8415

DHCPv6 Interaction (Simplified)



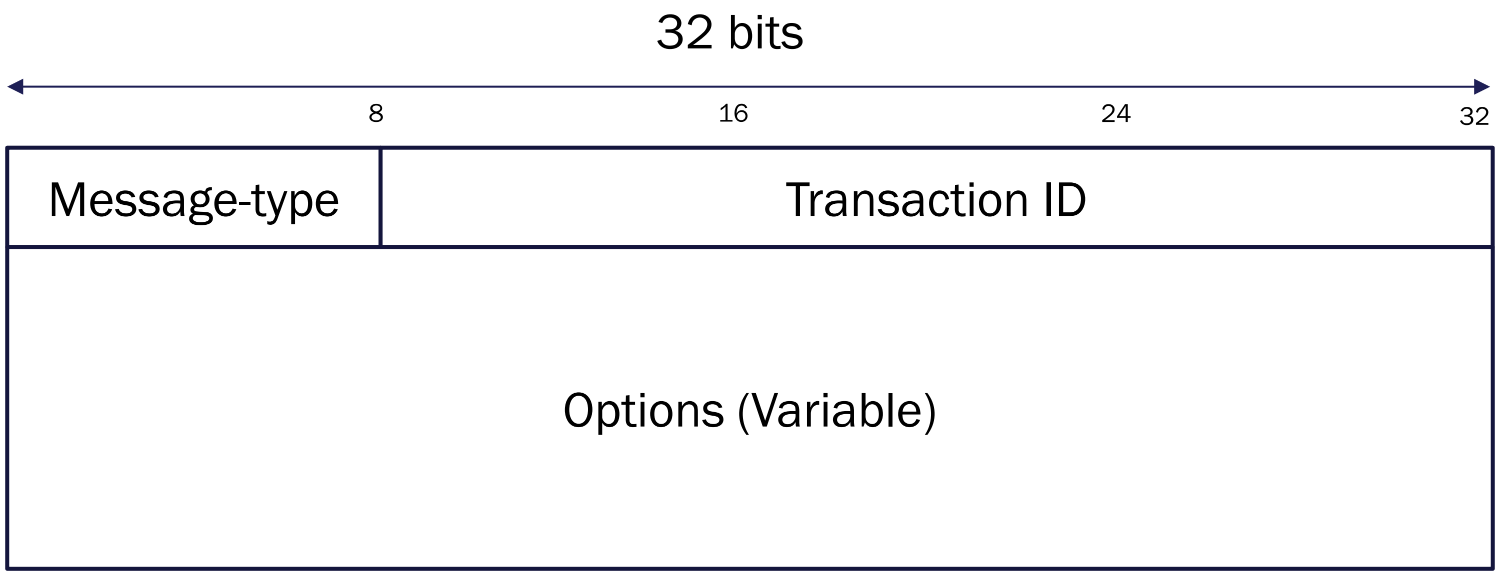
DHCP Request:
[fe80::2a0:24ff:fe71:e444]:546
sent to [ff02::1:2]:547

DHCP Response:
[fe80::2a0:24ff:fe71:0001]:547 ->
[fe80::2a0:24ff:fe71:e444]:546
IP Addres: a:c:7:9::3/64
Default gateway: a:c:7:9::1/64
Nameserver: a:c:7:9::10/64

DHCPv6 Message Format

- Client-Server messages are structured as below
- There can be many options
- No padding between options

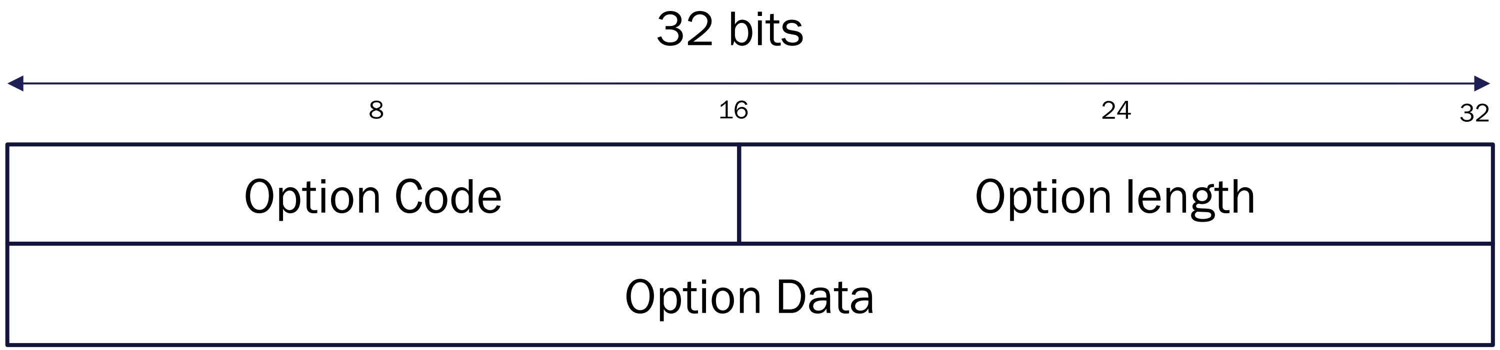
RFCs:
8415



BOOTP/DHCP Message Format *(continued)*

- Message Type: ie. SOLICIT, ADVERTISE…
- Transaction ID: value to synchronize server and client communications
  - Clients create random not easily guessed
- Options
  - Carry additional parameters in DHCP message

RFCs:
3315, 3319, 3736, 4776



32 bits

8	16	24	32

Option Code	Option length
Option Data	

DHCP Message Type

- DHCP Message types
- (RFC 3315 :: 5.3)

RFCs:
3315

VALUE	MESSAGE TYPE
1	SOLICIT
2	ADVERTISE
3	REQUEST
4	CONFIRM
5	RENEW
6	REBIND
7	REPLY
8	RELEASE

DHCP Message Type

- DHCP Message types
- (RFC 3315 :: 5.3)

RFCs:
3315

VALUE	MESSAGE TYPE
9	DECLINE
10	RECONFIGURE
11	INFORMATION-REQUEST
12	RELAY-FORW
13	RELAY-REPL

LINUX
CNO
Programming

ADVANCED CYBER TRAINING PROGRAM

HTTP

HyperText Transfer Protocol (HTTP)

- Communication protocol between clients/servers:
    - Client – requests, receives, and displays
    - Server – receives and responds to requests

RFCs:
IETF, W3C, 2616

Request

RFCs:
IETF, W3C, 2616

- Uniform Resource Locator (URL):
  - http://www.digg.com
  - https://www.yourbank.com
  - ftp://ftp.kernel.org
- Different URL schemes map to different services
- Hostname is converted from a name to a 32-bit IP address (DNS lookup, if needed)
- Connection is established to server (TCP)

Request Format

- Messages are in ASCII
- Carriage-return and line-feed indicate end of headers
- Headers may communicate private information
  - (browser, OS, cookie information, etc.)

RFCs:
IETF, W3C, 2616

```
GET / HTTP/1.1
Host: www.sourceforge.net
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.6) Gecko/2009011913 Firefox/3.0.6
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie:__utma=191645736.3937760829867591000.1233792450.1234402660.1234493716.3;...
```

Request Verbs

RFCs:
IETF, W3C, 2616

VERB	DESCRIPTION
GET	retrieve a file
HEAD	same as get but just meta-data
POST	submit data to be processed
PUT	upload a representation of the resource
DELETE	removed named resource
TRACE	“echo” for debugging
CONNECT	used by proxies for tunneling
OPTIONS	request for server/proxy options

Response Format

- Similar to request format:

```
HTTP/1.1 302 Found
X-Powered-By: PHP/5.2.6
X-SFX-Webhead: sfs-web-6
X-SFX-Revision: release_20090210.01
Location: http://sourceforge.net/index.php
Content-type: text/html
Content-Length: 0
Date: Fri, 13 Feb 2009 02:52:51 GMT
Server: lighttpd/1.4.19
```

RFCs:
IETF, W3C, 2616

Response Types

- 1XX: Informational
  - 100 Continue
  - 101 Switching Protocols
- 2XX: Success
  - 200 OK
  - 206 Partial Content
  - 207 Multi-Status
- 3XX: Redirection
  - 301 Moved Permanently
  - 302 Found
  - 304 Not Modified
- 4XX: Client error
  - 400 Bad Request
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not Found
- 5XX: Server error
  - 500 Internal Server Error
  - 503 Service Unavailable
  - 505 HTTP Version Not Supported

RFCs:
IETF, W3C, 2616

HyperText Transfer Protocol (HTTPS)

- HTTPS is a combination of HTTP and a network security protocol

RFCs:
2818

Operation

- HTTPS is not really a separate protocol
- Refers to the combination of normal HTTP interaction over an encrypted TLS or SSL connections
  - TLS runs between application protocols and above reliable transport protocols
- “HTTP/2 to only be used with https:// URIs on the "open" Internet. http:// URIs would continue to use HTTP/1”

RFCs:
2818

HTTP/2

- Released in May 2015 in RFC 7540
- Backwards compatible with HTTP 1.1
- Provides new functionality to support performance improvements
    - Data compression of HTTP headers
    - Server push
    - Pipelining of requests
    - Multiplexing of requests over single TCP connection

QUIC and HTTP/3

- **NOTE: HTTP/3 is still under development and not officially released. It is however supported by most browsers today**
- **Q**uick **U**DP **I**nternet **C**onnections or QUIC (quick) is a new transport layer protocol proposed by Google.
- The current plan for HTTP/3 is to switch from using TCP to using QUIC as the underlying transport layer protocol
- QUIC was designed to increase the speed of connections as well as make them more secure
- It essentially performs the equivalent of a 3-way TCP handshake as well as a TLS 1.3 handshake for all connections
- It solves the head-of-line blocking problem that HTTP/2 runs into

TLS/SSL History

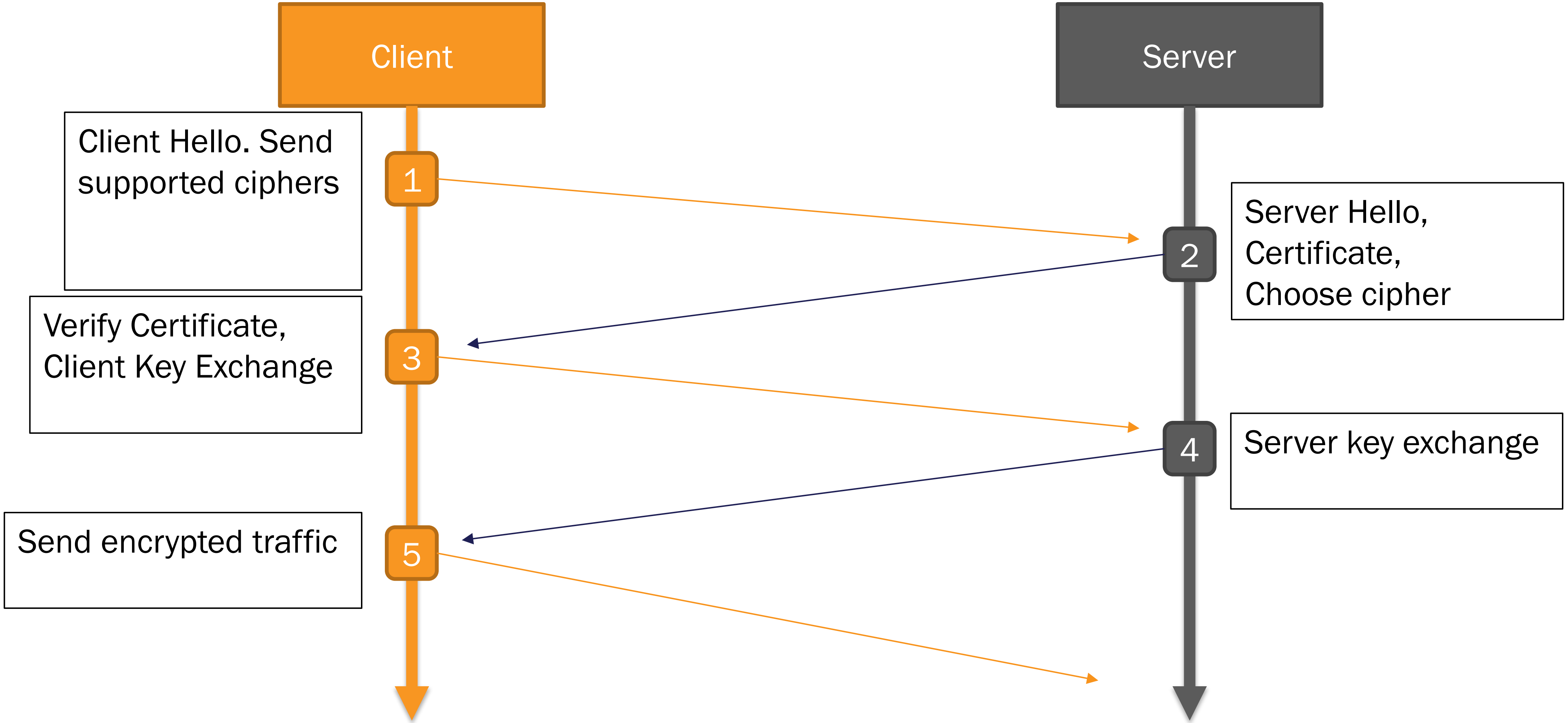
- SSL 1.0 was originally developed by Netscape but never publicly released
- 2.0 released 1995
- 3.0 released in 1996 and served as the basis for TLS 1.0, an IETF standard protocol defined in RFC 2246 in 1999
- TLS 1.0 released in 1999
- TLS 1.1 released in 2006 RFC 4346
- TLS 1.2 released in 2008 RFC 5246
- TLS 1.3 released in 2018 RFC 8446

TLS/SSL Implementation

- The client and server negotiate a connection and agree on various parameters to establish security:
  - Client connects to a TLS-enabled server and presents a list of supported ciphers and hash functions
  - Server picks the strongest of both that it supports and notifies the client
  - Server sends its identification in the form of a digital certificate containing the server name, the trusted CA, and the server's public key
  - Client checks server cert
  - Client encrypts a random number with the server's public key, and sends the result to the server
  - Both parties generate key material from the random number

RFCs:
2818

TLS 1.2 and earlier handshake



Client Hello. Send supported ciphers

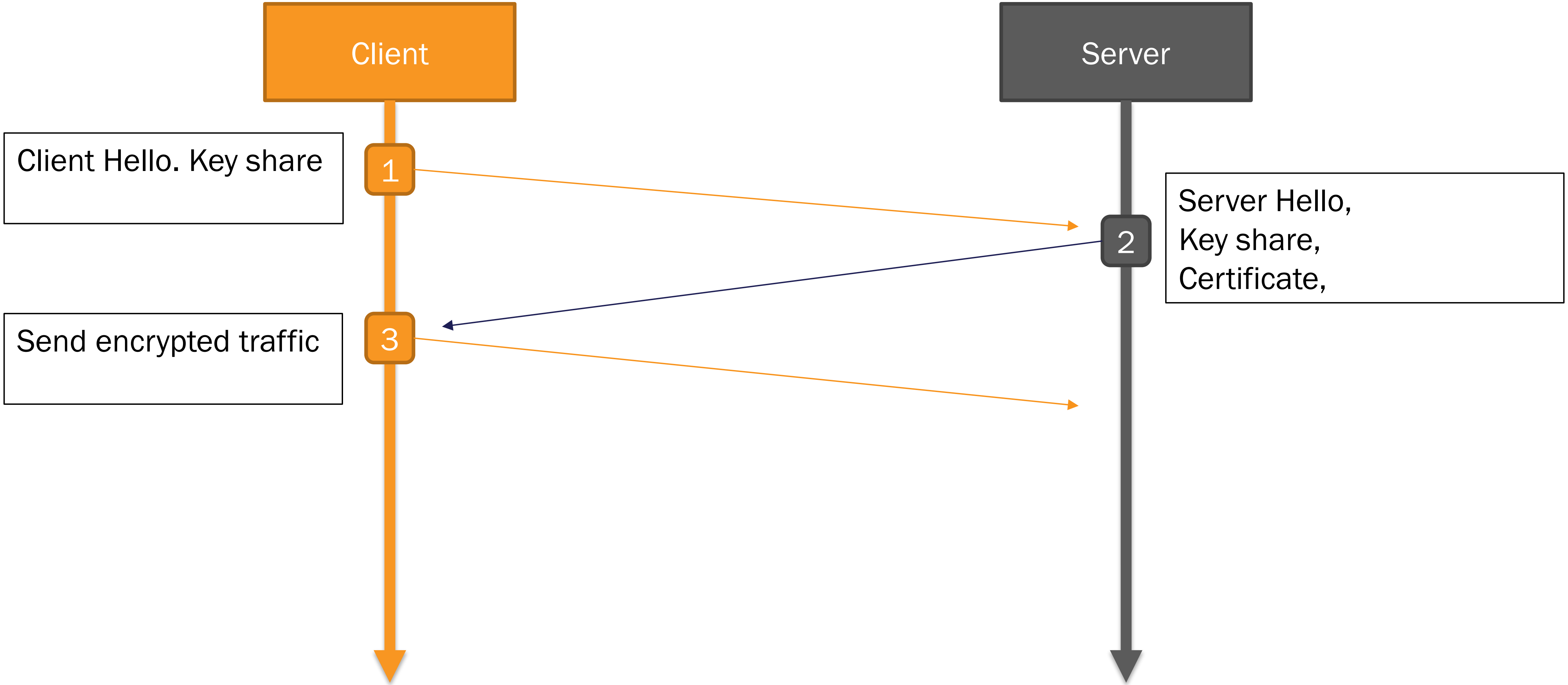
Server Hello, Certificate, Choose cipher

Verify Certificate, Client Key Exchange

Server key exchange

Send encrypted traffic

TLS 1.3 handshake



Client

Server

Client Hello. Key share

Server Hello,
Key share,
Certificate,

Send encrypted traffic

Demo Time!

- Feel free to follow along
- View a server’s certificate
  - https://canvas/
  - Take a look and discuss findings with class

LINUX
CNO
Programming

ADVANCED CYBER TRAINING PROGRAM

Lab 6

TLS sockets

TASK 1: HTTP/1.0 GET

- Set up a local webserver
  - Python has a built in webserver!
    - python2 –m SimpleHTTPServer $port
    - python3 –m http.server $port
- Connect to it in Firefox, and see what happens in Wireshark
- Use netcat to connect to the canvas.class.net:80
  - `ncat.exe canvas.class.net 80`
- Issue an HTTP/1.0 GET request for / using netcat. Pipe the output to a file

TASK 2: HTTP/1.1 GET

- Connect to webserver as in Task 2
- Issue an HTTP/1.1 GET request for /
- Refer to RFC 2616
- What is returned from server?
- Read the RFC on HTTP 1.0 (1945) and 1.1 (2616)
- What changed?
- Why? What key thing does 1.1 support that 1.0 did not?
    - The host-header field was added to support virtual hosts

TASK 3: Observation

- With Wireshark running…
- Use web browser to view https://canvas/
- What information is sent in the request?
- What information is returned by server?
- Can you ID each step in the SSL/TLS handshake?
- Which ciphers were offered by the client?
- Which cipher was chosen by the server?

Task 4: Write a TLS client and server

- Create a secure socket connection
  - First create a client that supports TLS. To verify it’s behavior have it try and connect to canvas
    - To do this you will need to specify the hostname. This can be found in canvas’s certificate
    - You will also need to add canvas’s self signed cert to the chain of trust. What python api call can do that?
  - Next try and create a server that supports TLS
    - For this you will need to generate a public and private key. Use the cmd in cmds.txt
  - For both of these be sure to view the traffic in wireshark to see that it is really encrypted

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

Domain Name System (DNS)

A hierarchical naming system for computers, services, or any resource participating in the Internet

DNS Introduction

- People prefer to use easy-to-remember names instead of IP addresses
- Domain names are alphanumeric names for IP addresses e.g., neon.ece.utoronto.ca, www.google.com, ietf.org
- The domain name system (DNS) is an Internet-wide distributed database that translates between domain names and IP addresses
- How important is DNS?
  - Imagine what happens when the local DNS server is down?

Google.com AAAA: 2a00:1450:8007::68

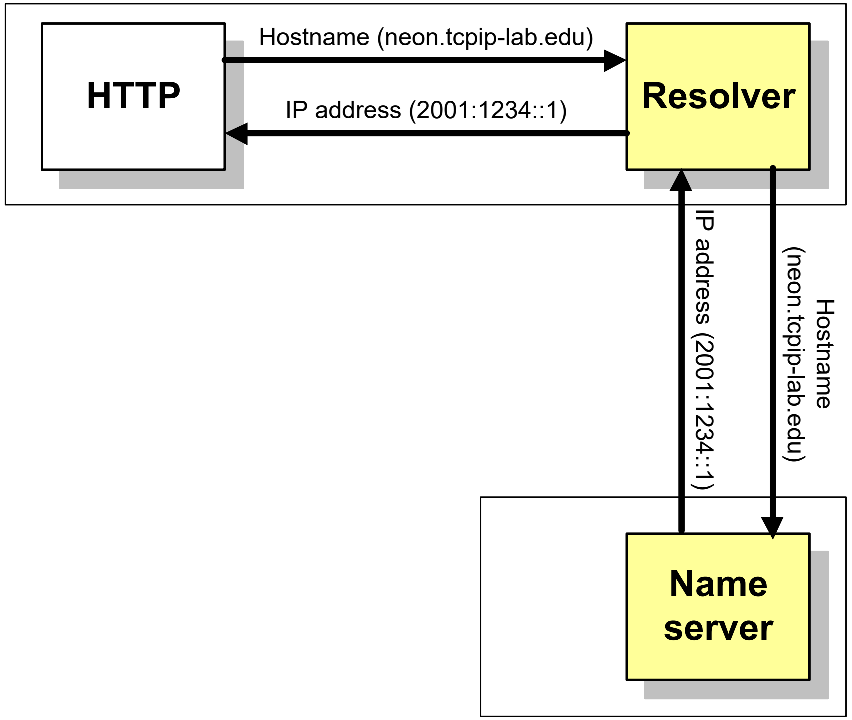
DNS Structure

1. An application program on a host accesses the domain system through a DNS client, called the resolver
2. Resolver contacts DNS server, called name server
3. DNS server returns IP address to resolver which passes the IP address to application

Reverse lookups are also possible, i.e., find the hostname given an IP address

RFCs:
1886

Resolver and Name Server



Design Principle of DNS

RFCs:
1886

- The naming system on which DNS is based is a hierarchical and logical tree structure called the domain namespace
- An organization obtains authority for parts of the name space, and can add additional layers of the hierarchy
- Names of hosts can be assigned without regard of location on a link layer network, IP network or autonomous system
- In practice, allocation of the domain names generally follows the allocation of IP address, e.g.,
  - All hosts with network prefix 2001:1234:1337::/64 have domain name suffix virginia.edu
  - All hosts on network 2001:1234:1338::/64 are in the Computer Science Department of the University of Virginia

Understanding Domains and Zones

- Domain is the namespace of an organization
- Domain virginia.edu can contain cs.virginia.edu and eng.virginia.edu
- A domain can have multiple name servers
- Domain Name servers store information about part of the domain space in “zones”
  - Virginia.edu would be considered one zone
  - cs.virginia.edu is another
  - eng.virginia.edu is a separate zone as well
- A name server can be authoritative over one zone or many

RFCs:
1886

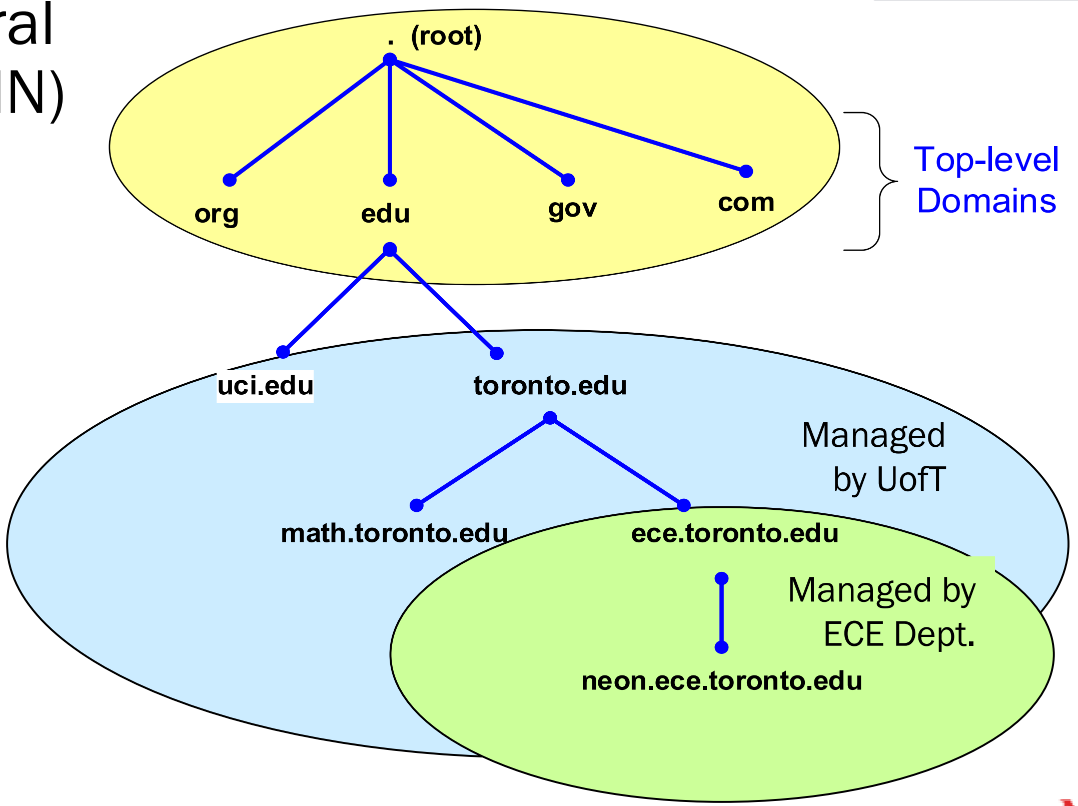
Hosts File

- Before DNS (until 1985), the name-to-IP address was done by downloading a single file (hosts.txt) from a central server with FTP
- Names in hosts.txt are not structured
- The hosts.txt file still works on most operating systems; it can be used to define local names
- Nowadays we have things like .io, .onion, .p2p, .biz, .info becoming popular

DNS Name Hierarchy

- DNS hierarchy can be represented by a tree
- Root and top-level domains are administered by an Internet central name registration authority (ICANN)
- Below top-level domain, administration of name space is delegated to organizations
- Each organization can delegate further

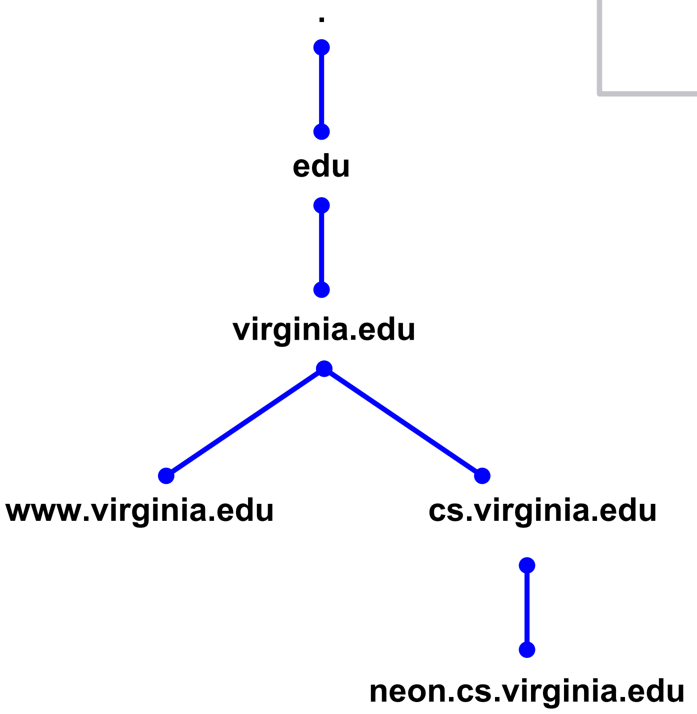
RFCs:
1035



Domain Name System

- Each node in the DNS tree represents a DNS name
- Each branch below a node is a DNS domain
- DNS domain can contain hosts or other domains (subdomains)
- Example: DNS domains are
    - .
    - Edu
    - virginia.edu
    - cs.virginia.edu

RFCs:
1034, 1035



Domain Names

RFCs:
1034, 1035

- Hosts and DNS domains are named based on their position in the domain tree
- Every node in the DNS domain tree can be identified by a unique Fully Qualified Domain Name (FQDN). The FQDN gives the position in the DNS tree
- A FQDN consists of labels (“cs”,“virginia”,”edu”) separated by a period (“.”)
- There can be a period (“.”) at the end
- Each label can be up to 63 characters long
- FQDN contains characters, numerals, and dash character (“-”)
- FQDNs are not case-sensitive

cs.virginia.edu

or

cs.virginia.edu.

Top-Level Domains

- Three types of top-level domains:
  - Organizational: 3-character code indicates the function of the organization
    - Used primarily within the US
    - Examples: gov, mil, edu, org, com, net
  - Geographical: 2-character country or region code
    - Examples: us, va, jp, de
  - Reverse domains: A special domain (in-addr.arpa) used for IP address-to-name mapping
- There are more than 1543 top-level domains
- In 2011 ICANN approved expansion of TLDs to any extension; estimated fee: $185,000

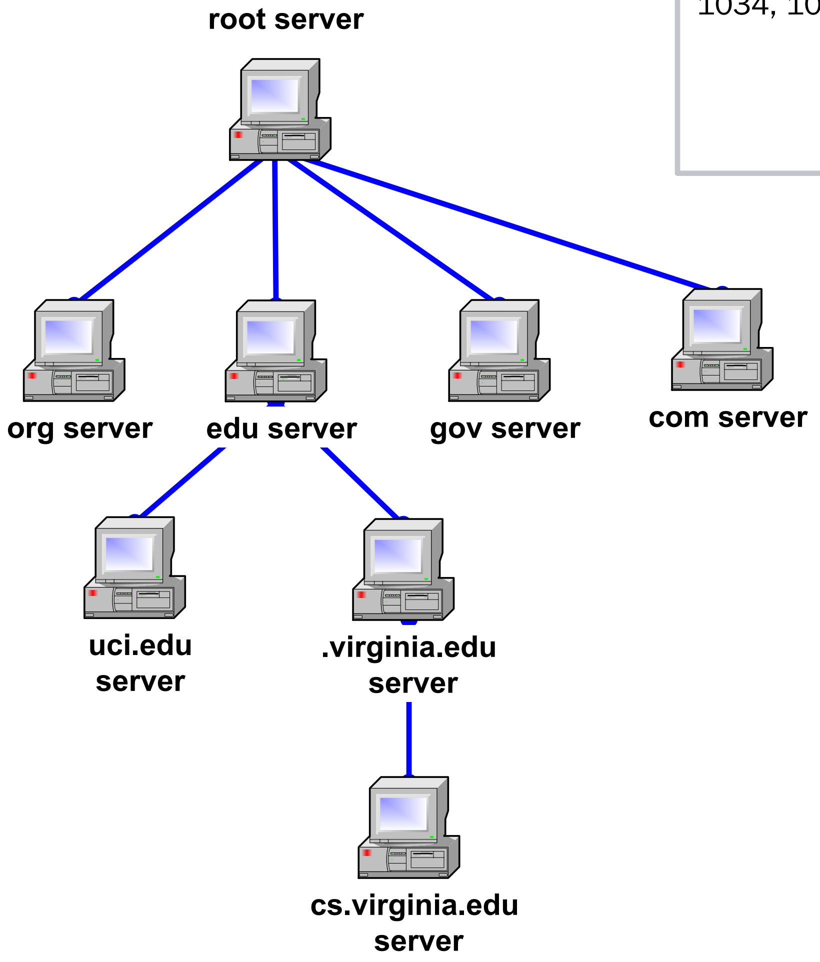
Organizational TLD’s

com	Commercial organizations
edu	Educational institutions
gov	Government institutions
int	International organizations
mil	U.S. military institutions
net	Networking organizations
org	Non-profit organizations

Hierarchy of Name Servers

- The resolution of the hierarchical name space is done by a hierarchy of name servers
- Each server is responsible (authoritative) for a contiguous portion of the DNS namespace, called a zone
- Zone is a part of the subtree
- DNS server answers queries about hosts in its zone

RFCs:
1034, 1035



Authority and Delegation

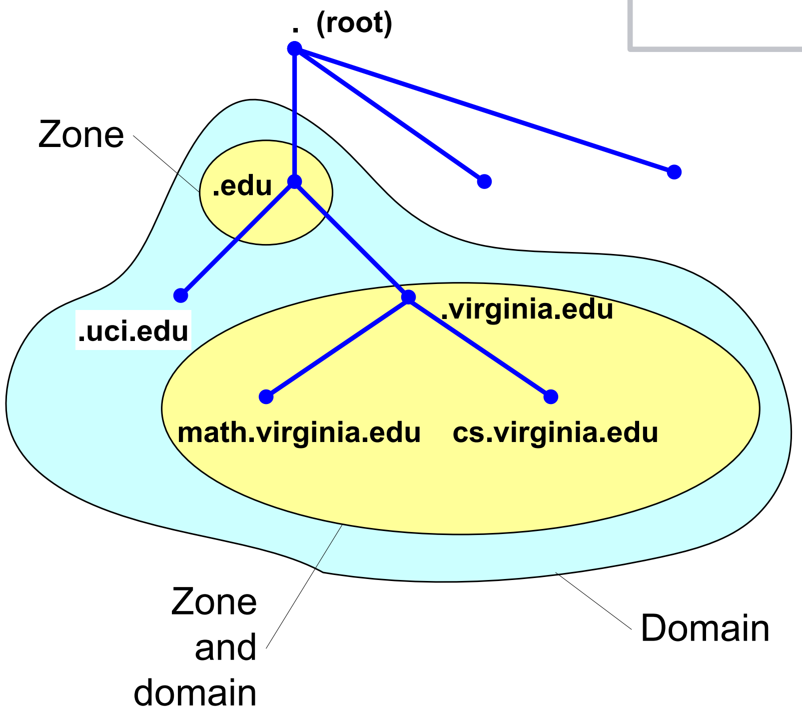
- Authority for the root domain is with the Internet Corporation for Assigned Numbers and Names (ICANN)
- ICANN delegates to accredited registrars (for gTLDs) and countries for country code top level domains (ccTLDs)
- Authority can be delegated further
- Chain of delegation can be obtained by reading domain name from right to left
- Unit of delegation is a “zone”

RFCs:
1034, 1035

DNS domains and Zones

- Each zone is anchored at a specific domain node, but zones are not domains
- A DNS *domain* is a branch of the namespace
- A zone is a portion of the DNS namespace generally stored in a file (It could consists of multiple nodes)
- A server can divide part of its zone and delegate it to other servers

RFCs:
1034, 1035



Primary and Secondary Name Servers

RFCs:
1034, 1035

- For each zone, there must be a primary name server and a secondary name server
  - The primary server (master server)
    - Maintains a zone file which has information about the zone
    - Updates are made to the primary server
  - The secondary server copies data stored at the primary server
- When a new host is added (“gold.cs.virginia.edu”) to a zone, the administrator adds the IP information on the host (IP address and name) to a configuration file on the primary server

Root Name Servers

- The root name servers know how to find the authoritative name servers for all top-level zones
- There are 13 root name servers
  - The root server system has 1,844 instances operated by 12 independent organizations
- Root servers are critical for the proper functioning of name resolution

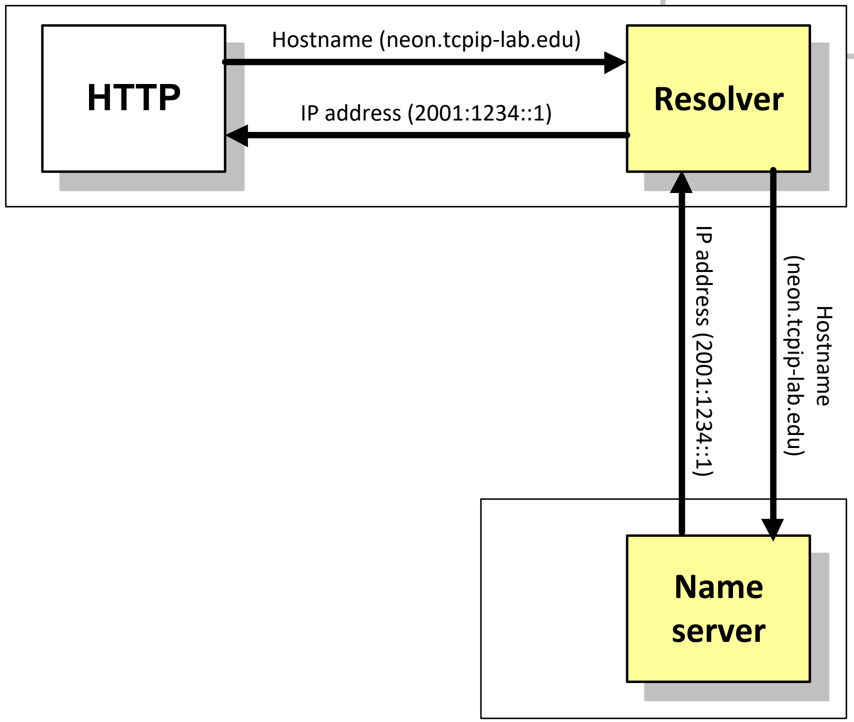
Addresses of Root Servers

SERVER	OPERATOR	IP ADDRESS
A	VeriSign, Inc.	198.41.0.4
B	Information Sciences Institute	192.228.79.201
C	Cogent Communications	192.33.4.12
D	University of Maryland	128.8.10.90
E	NASA Ames Research Center	192.203.230.10
F	Internet Systems Consortium, Inc.	192.5.5.241
G	U.S. DOD Network Information Center	192.112.36.4
H	U.S. Army Research Lab	128.63.2.53
I	Autonomica	192.36.148.17
J	VeriSign, Inc.	192.58.128.30
K	RIPE NCC	193.0.14.129
L	ICANN	199.7.83.42
M	WIDE Project	202.12.27.33

DNS Queries

- User program issues a request for the IP address of a hostname
- Local resolver formulates a DNS query to the name server of the host
- Name server checks if it is authorized to answer the query.
    - If yes, it responds
    - Otherwise, it will query other name servers, starting at the root tree
- When the name server has the answer it sends it to the resolver

RFCs:
1034, 1035



Recursive and Iterative Queries

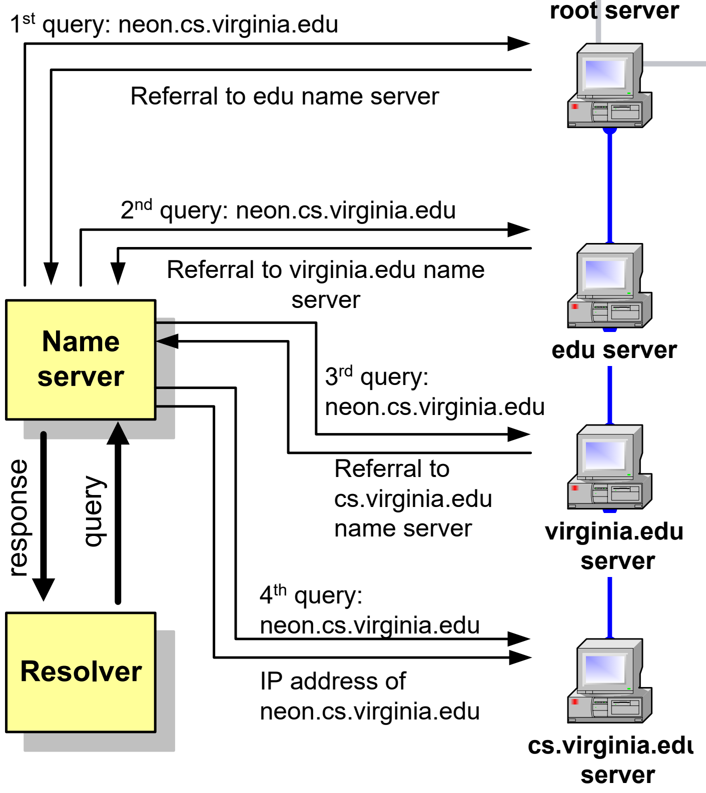
- There are two types of queries:
  1. Recursive queries: When the name server of a host cannot resolve a query, the server issues a query to resolve the query
  2. Iterative (non-recursive) queries: When the name server of a host cannot resolve a query, it sends a referral to another server to the resolver
- The type of query is determined by a bit in the DNS query

RFCs:
1034, 1035

Recursive Queries

- In a recursive query, the resolver expects the response from the name server
- If the server cannot supply the answer, it will send the query to the “closest known” authoritative name server (here: In the worst case, the closest known server is the root server)
- The root sever sends a referral to the “edu” server
- Querying this server yields a referral to the server of “virginia.edu”
- ... and so on

RFCs:
1034, 1035



1st query: neon.cs.virginia.edu

Referral to edu name server

2nd query: neon.cs.virginia.edu

Referral to virginia.edu name server

3rd query: neon.cs.virginia.edu

Referral to cs.virginia.edu name server

4th query: neon.cs.virginia.edu

IP address of neon.cs.virginia.edu

root server

edu server

virginia.edu server

cs.virginia.edu server

Name server

Resolver

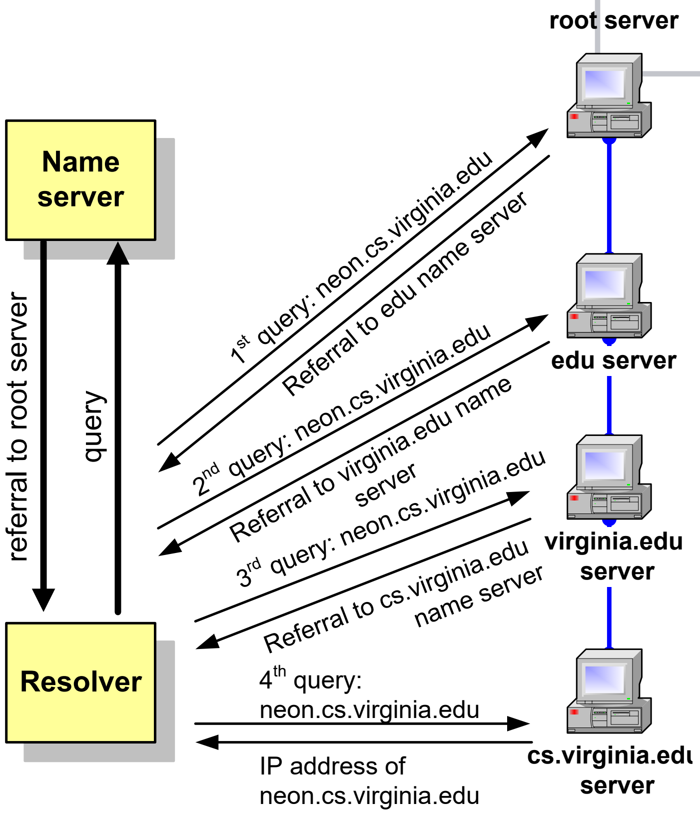
response

query

Iterative Queries

- In an iterative query, the name server sends a closest known authoritative name server a referral to the root server.
- This involves more work for the resolver

RFCs:
1034, 1035



root server

edu server

virginia.edu server

cs.virginia.edu server

Name server

Resolver

query

referral to root server

1st query: neon.cs.virginia.edu

Referral to edu name server

2nd query: neon.cs.virginia.edu

Referral to virginia.edu name server

3rd query: neon.cs.virginia.edu

Referral to cs.virginia.edu name server

4th query: neon.cs.virginia.edu

IP address of neon.cs.virginia.edu

Caching

- To reduce DNS traffic, name servers cache information on domain name/IP address mappings
- When an entry for a query is in the cache, the server does not contact other servers
- Note: If an entry is sent from a cache, the reply from the server is marked as "unauthoritative"

RFCs:
1034, 1035

Resource Records

db.mylab.com

```
$TTL 86400

mylab.com. IN SOA PC4.mylab.com.

               hostmaster.mylab.com. (

                              1 ; serial

                              28800 ; refresh

                              7200 ; retry

                              604800 ; expire

                              86400 ; ttl

                              )

;

mylab.com.     IN             NS             PC4.mylab.com.

;

localhost                IN   AAAA           ::1

PC4.mylab.com.           IN   AAAA           2001:800::4

PC3.mylab.com.           IN   AAAA           2001:800::3

PC2.mylab.com.           IN   AAAA           2001:800::2

PC1.mylab.com.           IN   AAAA           2001:800::1
```

- The database records of the distributed data base are called resource records (RR)
- Resource records are stored in configuration files (zone files) at name servers
- Left Resource records for a zone:

RFCs:
1035, 1912

Resource Records

db.mylab.com

```
$TTL 86400
mylab.com. IN SOA PC4.mylab.com.
                  hostmaster.mylab.com. (
                                  1 ; serial
                                  28800 ; refresh
                                  7200 ; retry
                                  604800 ; expire
                                  86400 ; ttl
                                  )
;
mylab.com.      IN                NS              PC4.mylab.com.
;
localhost                   IN    AAAA            ::1
PC4.mylab.com.              IN    AAAA            2001:800::4
PC3.mylab.com.              IN    AAAA            2001:800::3
PC2.mylab.com.              IN    AAAA            2001:800::2
PC1.mylab.com.              IN    AAAA            2001:800::1
```

Max. age of cached data
in seconds

* Start of authority (SOA) record.
Means: “This name server is authoritative for the zone Mylab.com”
* PC4.mylab.com is the name server
* hostmaster@mylab.com is the email address of the person in charge

Name server (NS) record.
One entry for each authoritative name server

Address (AAAA) records.
One entry for each hostaddress

RFCs:
1035, 1912

LINUX
CNO
Programming

ACTP
ADVANCED CYBER TRAINING PROGRAM

Lab 7

DNS

Task 0: View dns queries in wireshark

- Open wireshark and start sniffing
- Run nslookup for some site. What does the dns packet look like?

RFCs:
1034, 1035

TASK 1: AAAA & CNAME Records

- With sniffer running…
  - Start nslookup
  - Issue queries for the boxes you’ve used in class so far
    - linux.share.class.net
    - canvas.class.net
  - Inspect request/response in capture
  - Observe output in nslookup
    - Now run “set deb” inside nslookup and repeat queries
  - What information is returned?
  - What if there are multiple addresses returned?

TASK 2: PTR Records

- With sniffer running…
  - Start nslookup
  - Issue queries for the given IP addresses
    - 1.1.1.1
  - Inspect request/response in capture
  - Observe output in nslookup
  - What information can be gleaned from doing this?

TASK 3: NS Records

- With sniffer running…
  - Start nslookup
  - Set type to NS
    - `set type=ns`
  - Issue queries for the given names:
    - good.com
  - Inspect request/response in capture
  - Observe output in nslookup
  - What information is returned?

TASK 4: MX Records

- With sniffer running…
    - Start nslookup
    - Set type to MX
        - `set type=mx`
    - Issue queries for the given names
        - good.com
    - Inspect request/response in capture
    - Observe output in nslookup
    - What information is returned?
    - What does the preference value mean?

TASK 5: SOA Records

- With sniffer running…
  - Start nslookup
  - Set type to SOA (Start of Authority)
    - `set type=soa`
  - Issue queries for the given names
    - evil.com
  - Inspect request/response in capture
  - Observe output in nslookup
  - What information is returned?
  - Did you ever really go to the real “evil.com”?