CS 114:
Network Security

Lecture 2

Prof. Daniel Votipka
Spring 2023

(some slides courtesy of Prof. Micah Sherr and Prof. Bill Nace)

Tufts
UNIVERSITY

Administrivia

- Homework 0 due Jan. 26th (Thursday) at 11:59pm
- Homework 1, part 1 due Feb. 2nd at 11:59pm
  - Make sure close sockets on CTRL-C
  - There are no additional tests
- If you recently joined the class and do not have access to Gradescope, Piazza, and Box, please email me (daniel.votipka@tufts.edu) or come talk to me after class.

The Plan

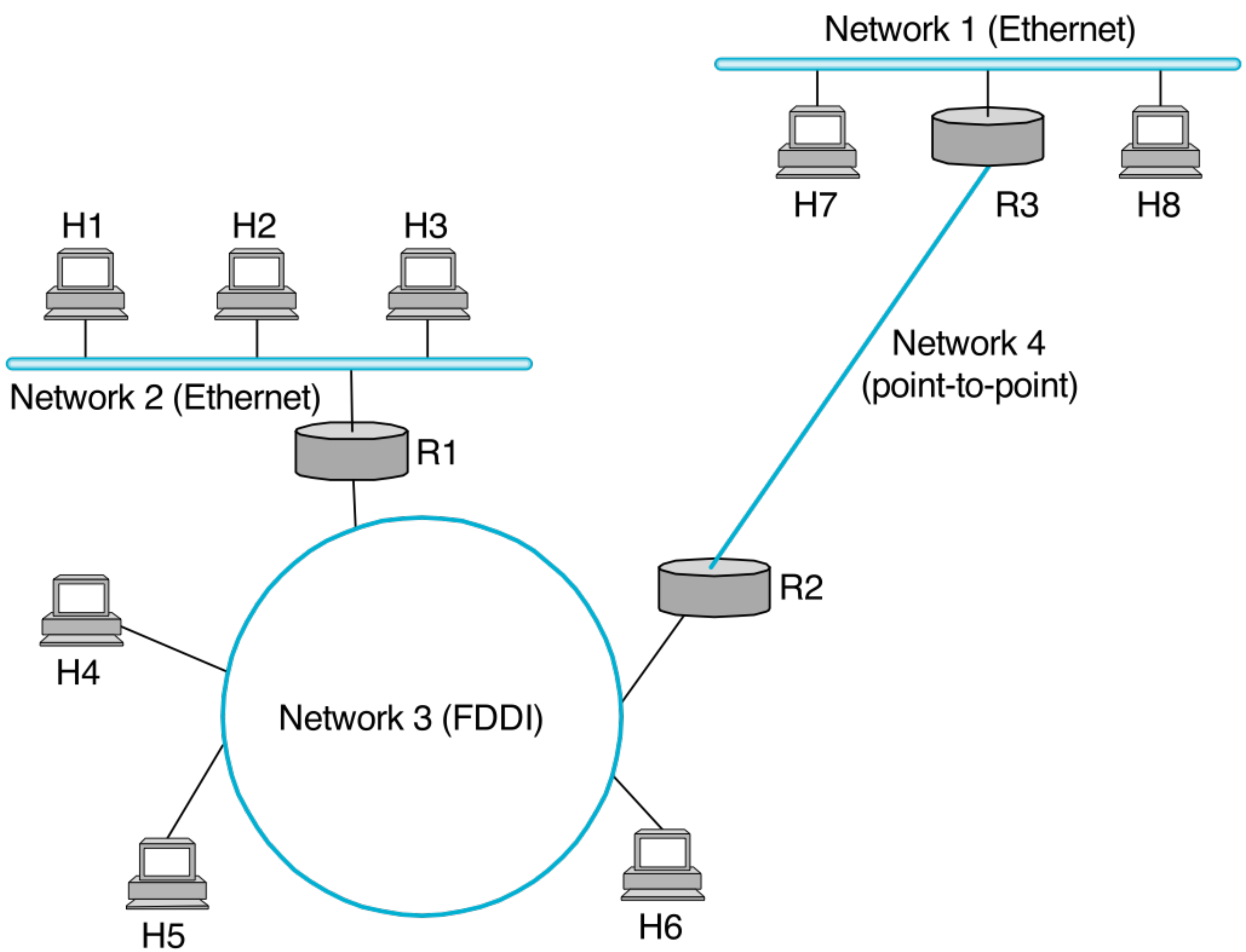
- Introduction to Networking
- The OSI Model
- Network Programming

Intro to Networking

Internet History 101

- DARPA – Defense Advanced Research Projects Agency
- ARPANET:
  - World's first operational packet switching network
  - Predecessor of global Internet
  - Started in 1969 with 4 routers @ UCLA, Stanford, UC Santa Barbara, Utah
  - TCP/IP in 1983

Fundamental Goal: An Inter-network

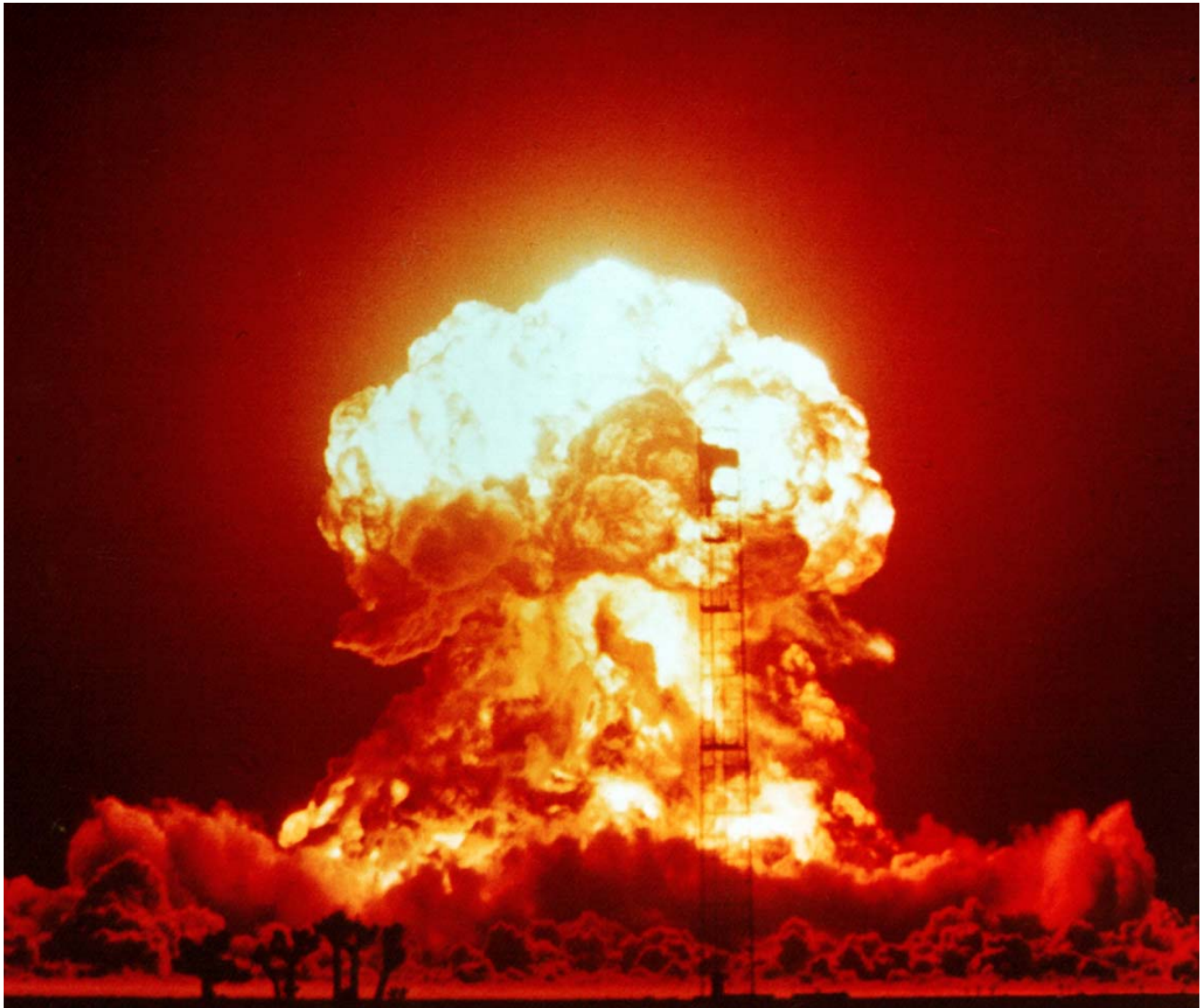


Hn = host, Rn = router/gateway

Goals of the Internet

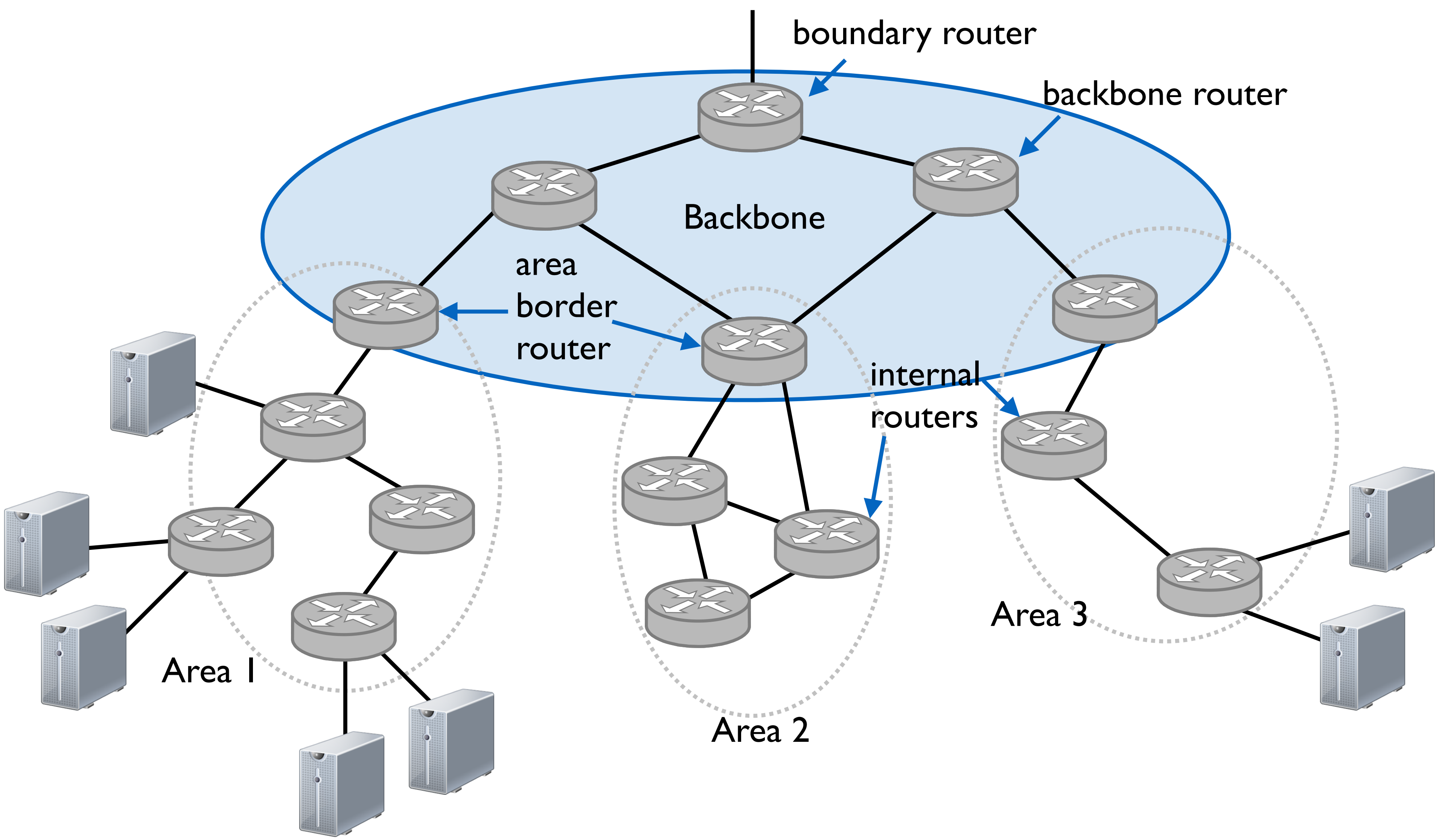
- Fundamental goal: Inter-connect multiple networks of different types (wired and wireless) via store-and-forward gateways
- Second-level goals:
  - Robust in face of failures
  - Support multiple types of services
  - Support a variety of networks
  - Allow distributed management
  - Cost effective

Not a major goal:
Route around nuclear explosion



What is the Internet?

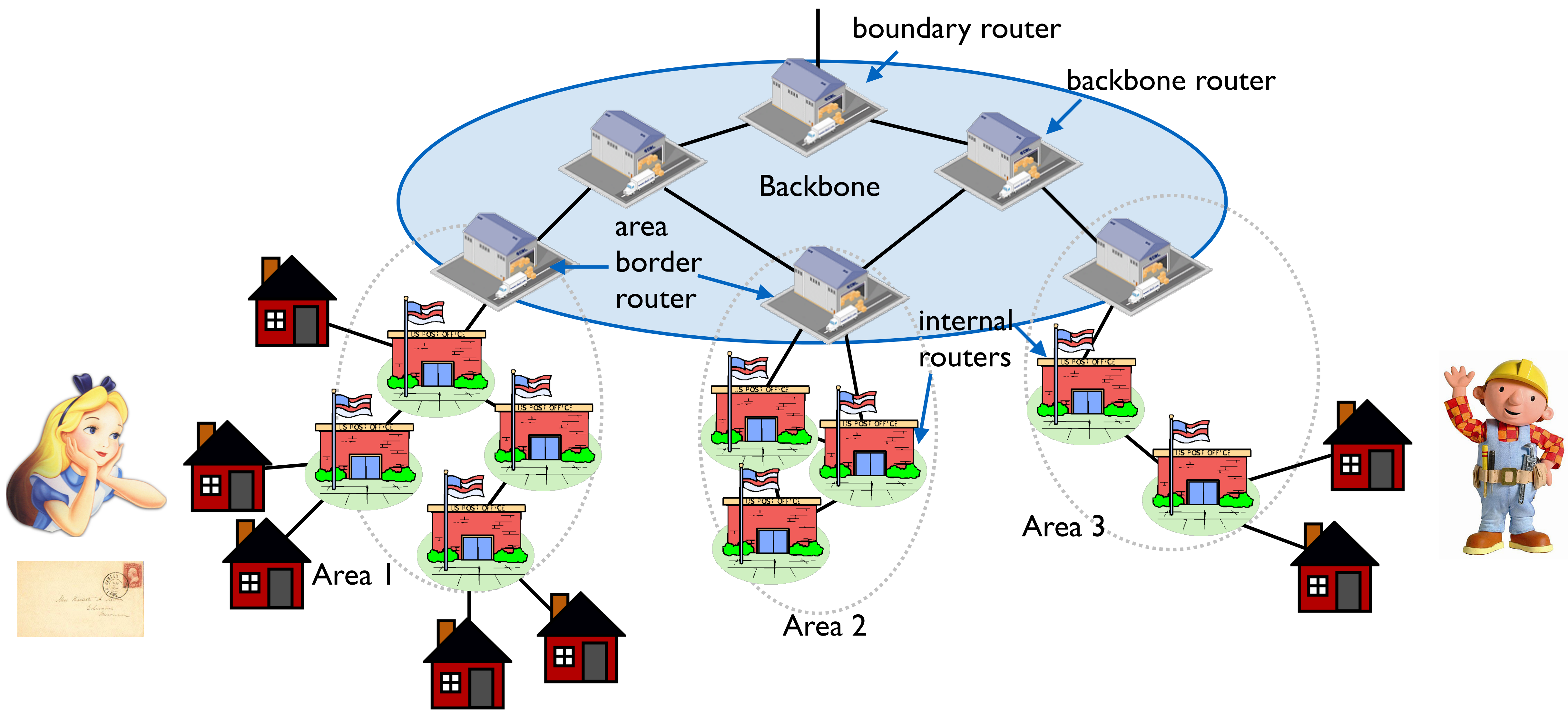
A collection of independently operated
autonomous systems (ASes)



What is the Internet?

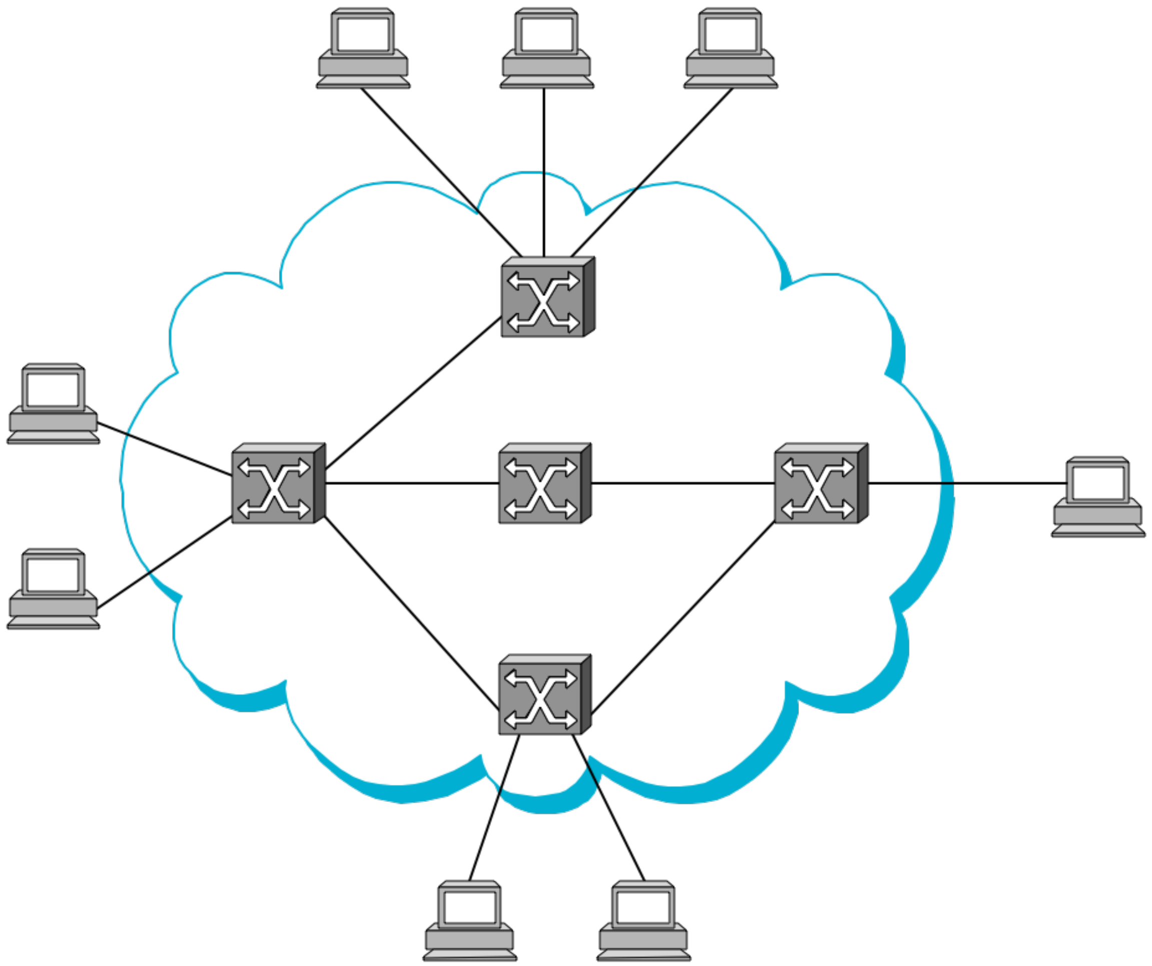
What is the Internet?

A collection of independently operated *autonomous systems (ASes)*



Switched Network

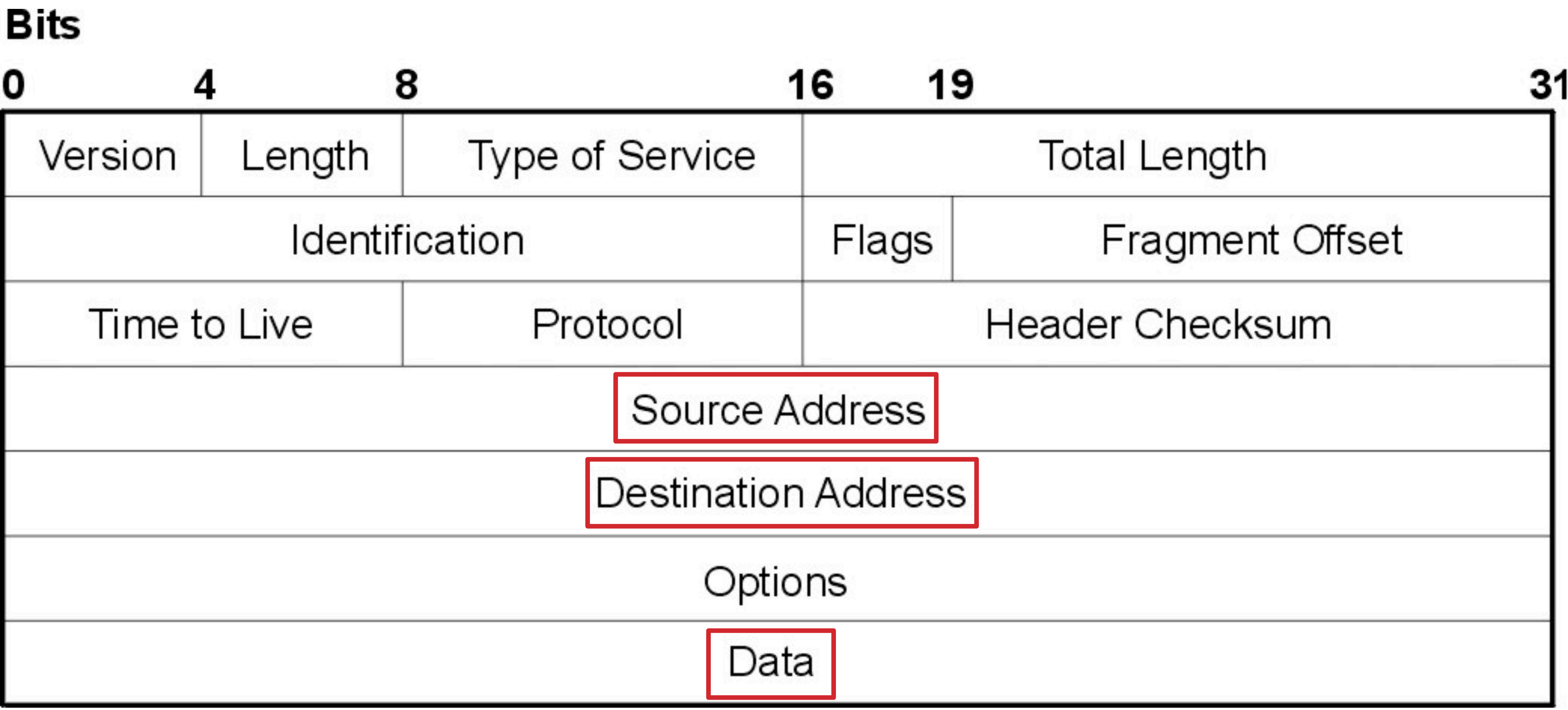
- End-hosts connected to switches
- **Switches:**
  - Forwarding nodes
  - At least two links
  - Also known as bridges or routers



Datagram Packet Switching

- **Packets** – discrete blocks of data
    - Each packet is independently switched
    - Each packet header contains destination address
    - Routing protocol is used to compute next hop
- Example: IP networks
- Advantages:
    - No connection state required
    - Easy to recover from errors
    - Minimal network assumptions

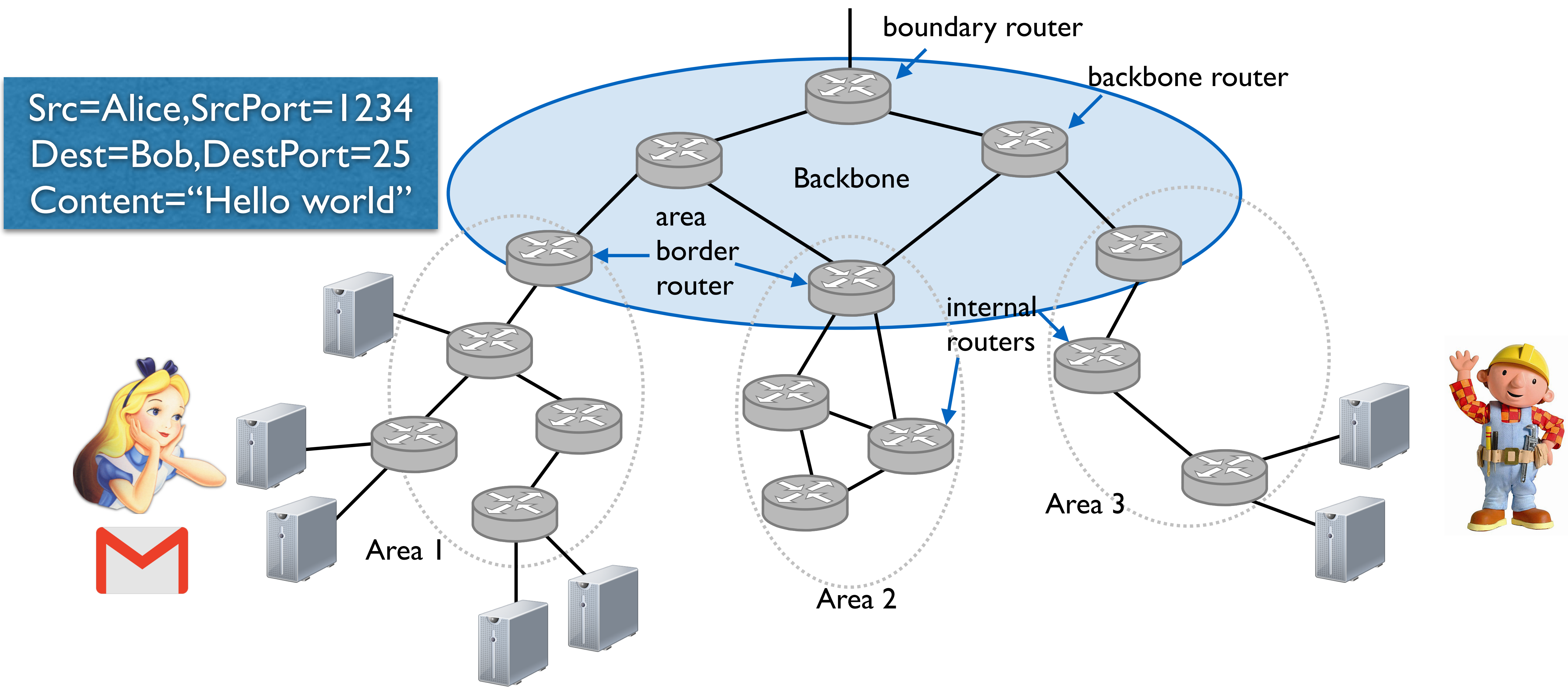
IP Packet Structure



https://tools.ietf.org/html/rfc791

What is the Internet?

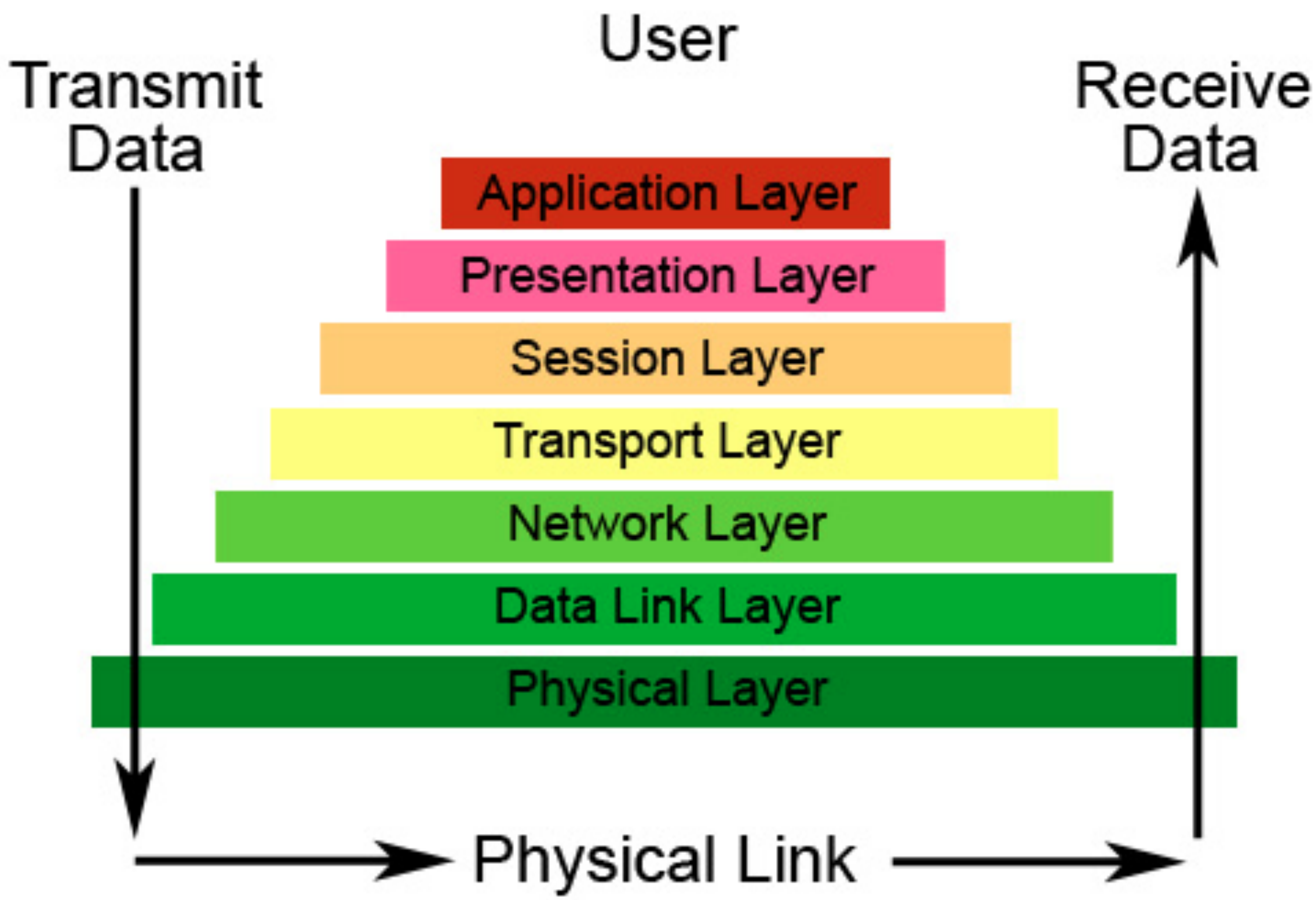
A collection of independently operated *autonomous systems (ASes)*



The structure of the Internet: The ISO/OSI Architecture

- ISO/OSI architecture describes functions of communication systems
  - ISO: International Standards Organization
  - OSI: Open System Interface
- Breaks communication systems into **layers**, with each layer serving as an *abstraction* to layer above

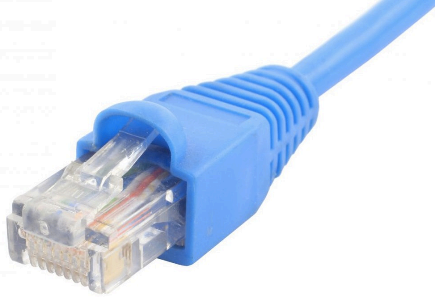
The Seven Layers of OSI



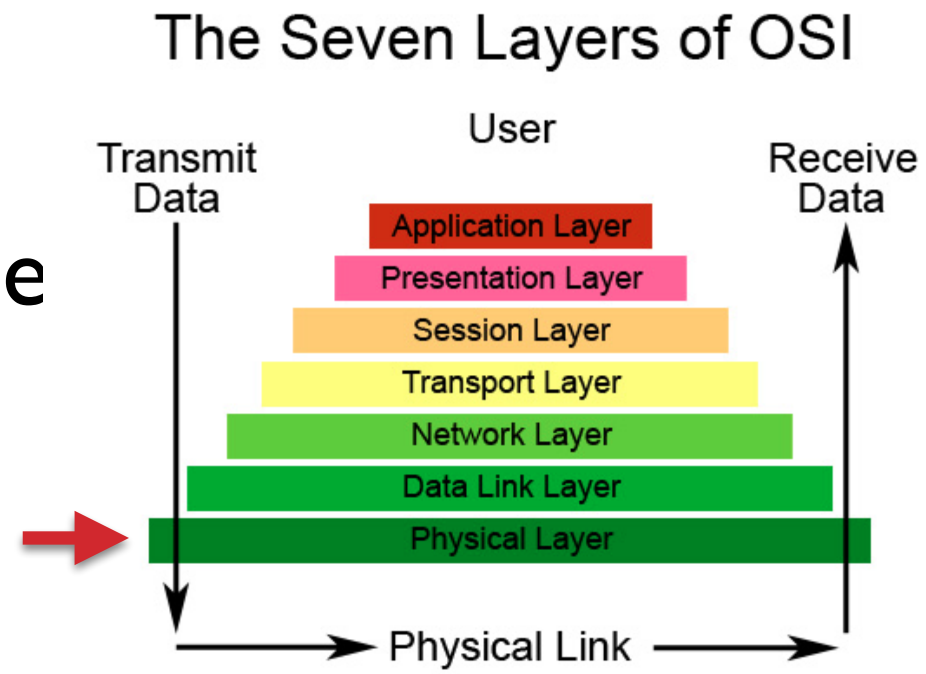
Three Properties of Layers

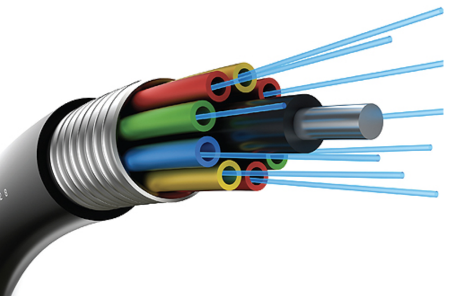
- **Service**: what a layer does
- **Service interface**: how to access the service (interface for layer above)
- **Protocol**: set of rules and formats that govern the communication between two network boxes

Physical Layer (1)

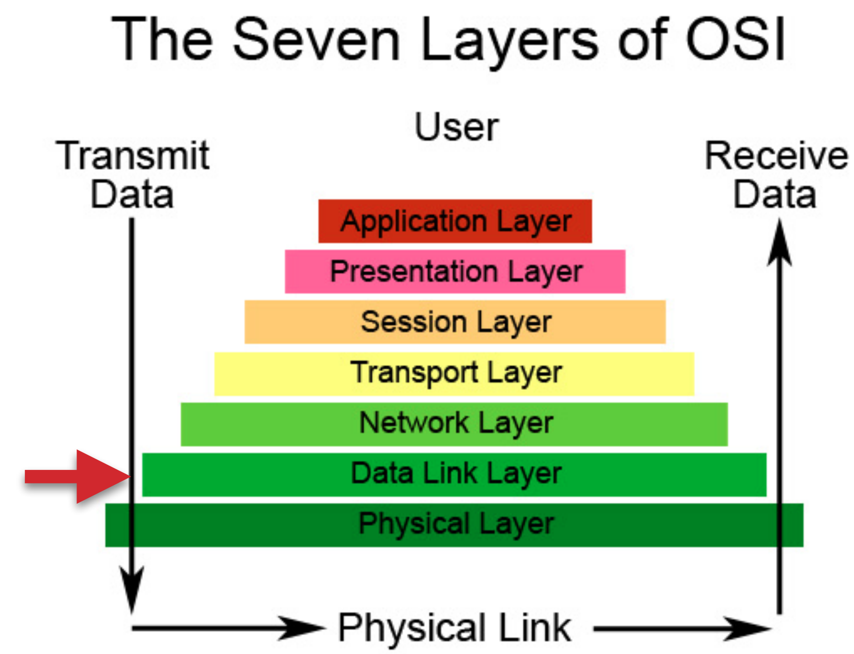


- Service: move information between two systems connected by a physical link
- Interface: specifies how to send a bit
- Protocol: coding scheme used to represent a bit, voltage levels, duration of a bit
- Examples: coaxial cable, optical fibe links.



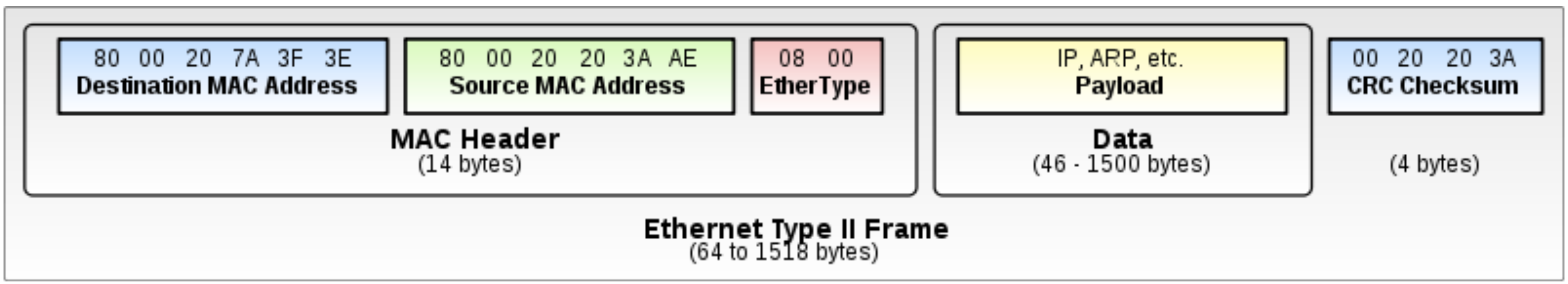


Datalink Layer (2)

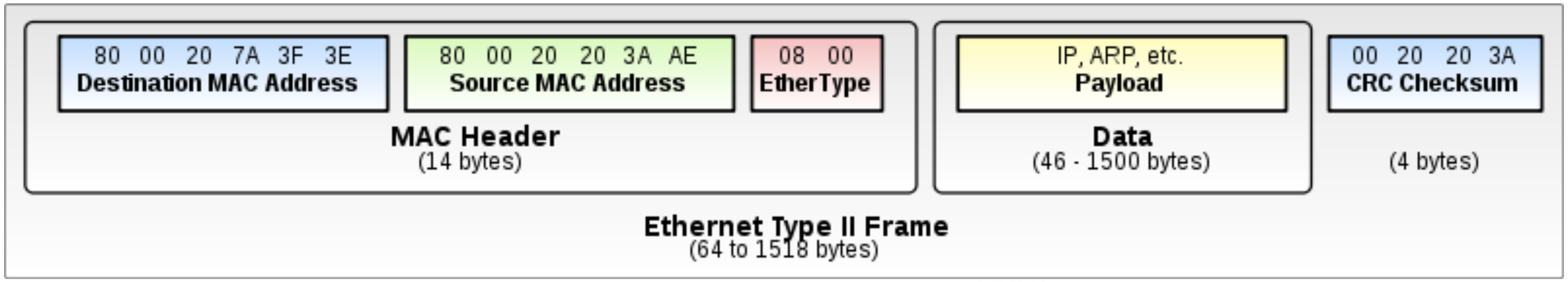


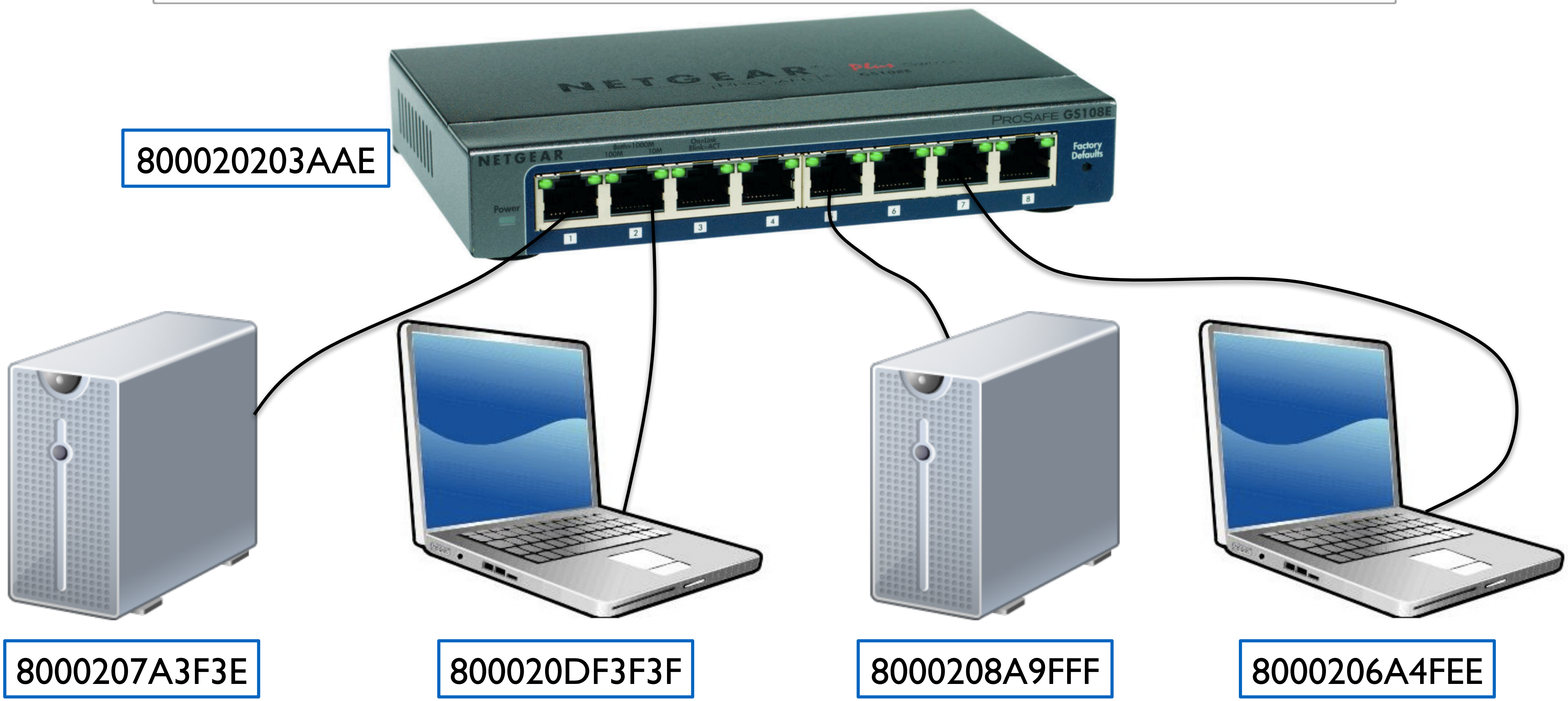
- Service:
  - Combine stream of bits into **frames** (attach frame separators)
  - Send data frames between peers
- Interface: send a data unit (frame) to a machine (MAC address) connected to the same physical media
- Protocol: Medium Access Control (MAC)
  - Examples: Ethernet (LAN), 802.11 (wireless)

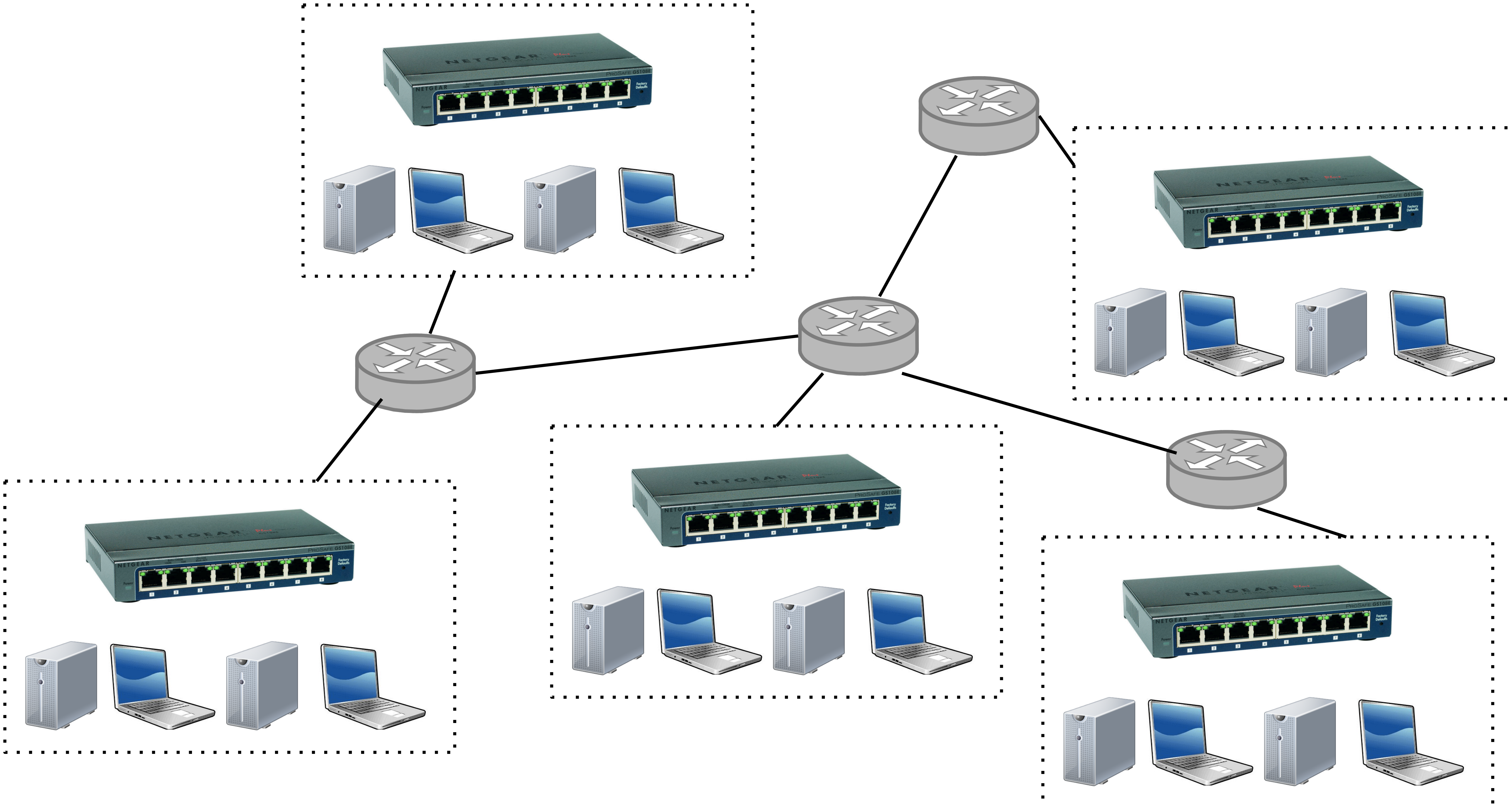
Ethernet Frame



An Ethernet Network

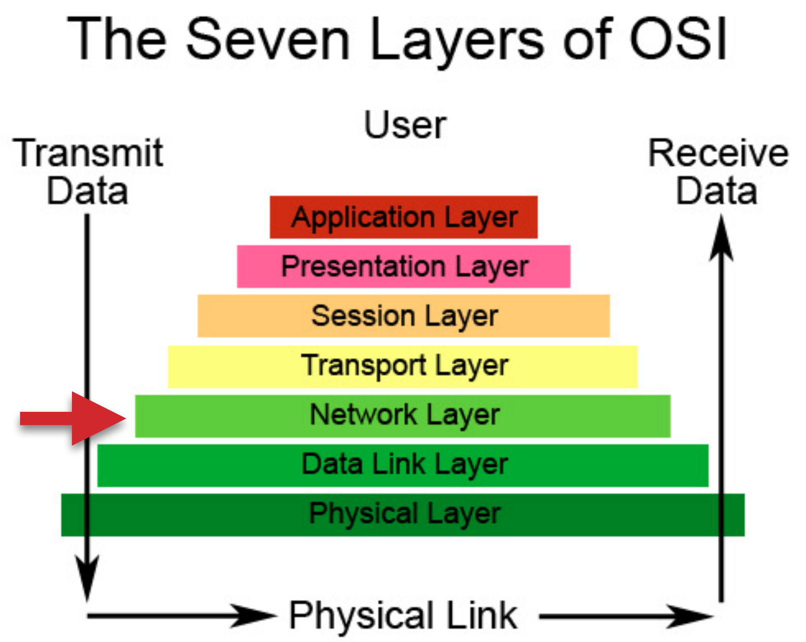






Network Layer (3)

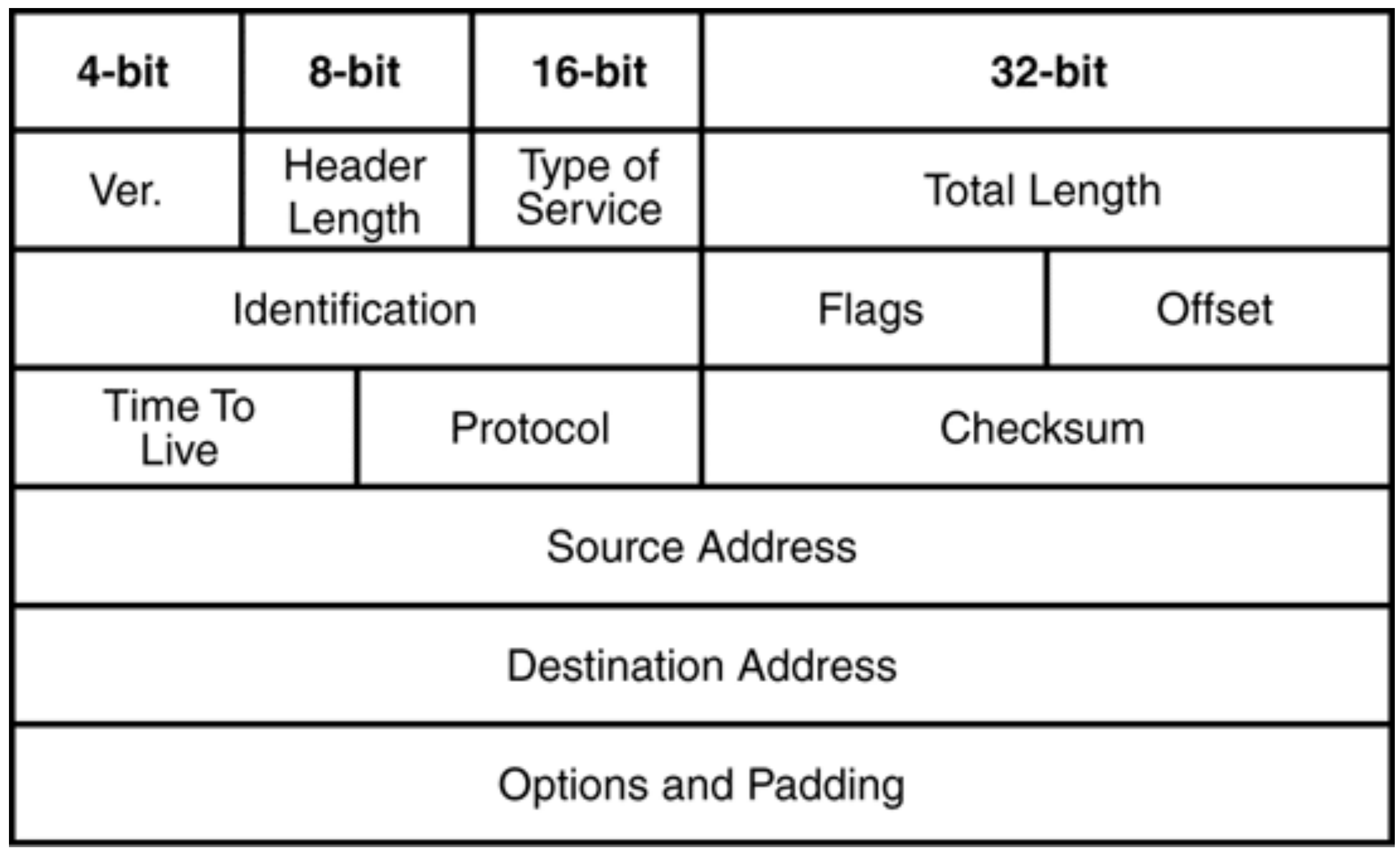
- Service:
  - Deliver a packet to specified network destination
  - Perform segmentation/reassembly
- Interface: send a packet to a specified destination (network address)
- Protocol: define global unique addresses; construct routing tables
  - Example: IP



Internet Protocol (IP)

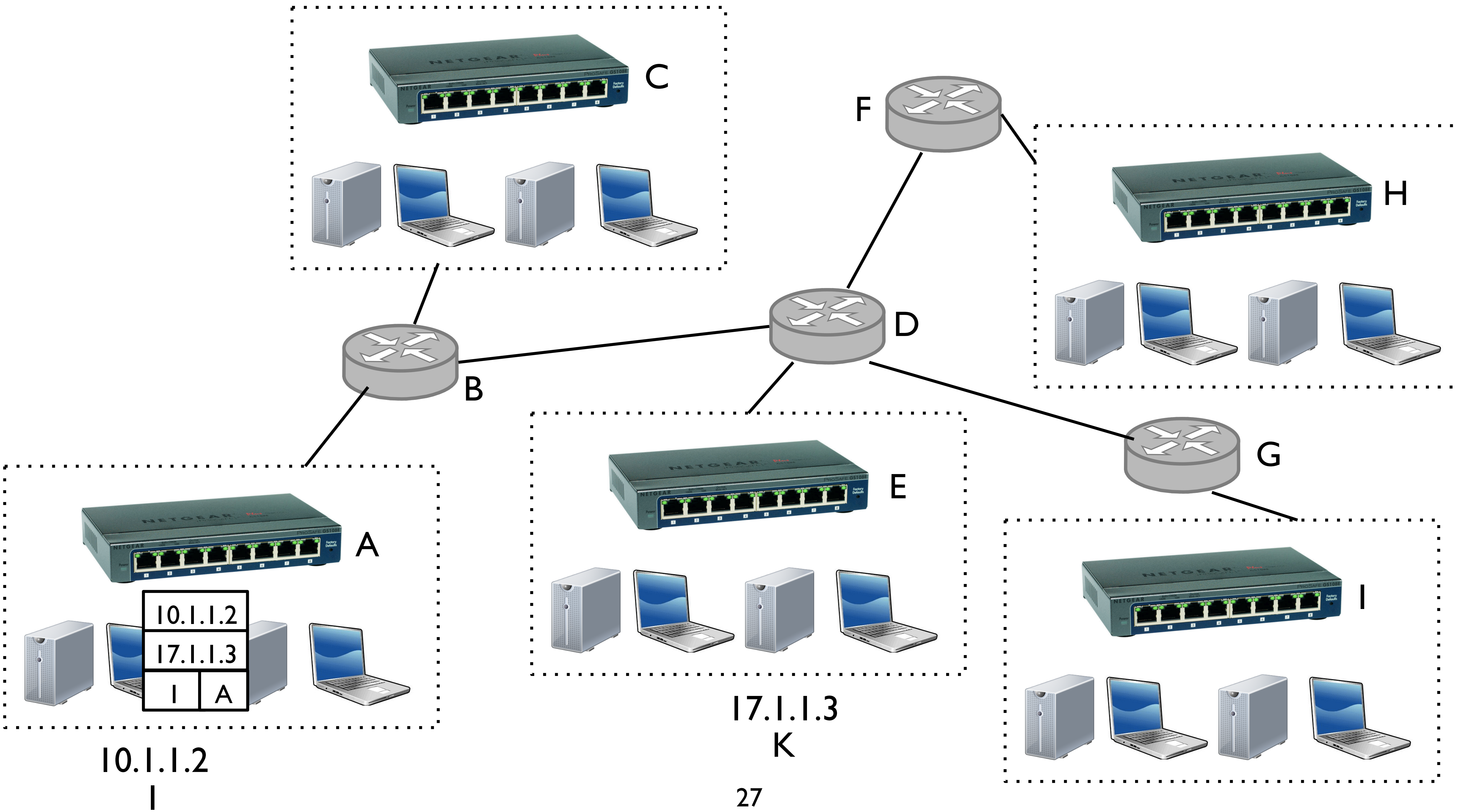
- Allows networks to interoperate
  - Any network technology that supports IP can exchange packets
- Allows applications to function on all networks
  - Applications that can run on IP can use any network

IP

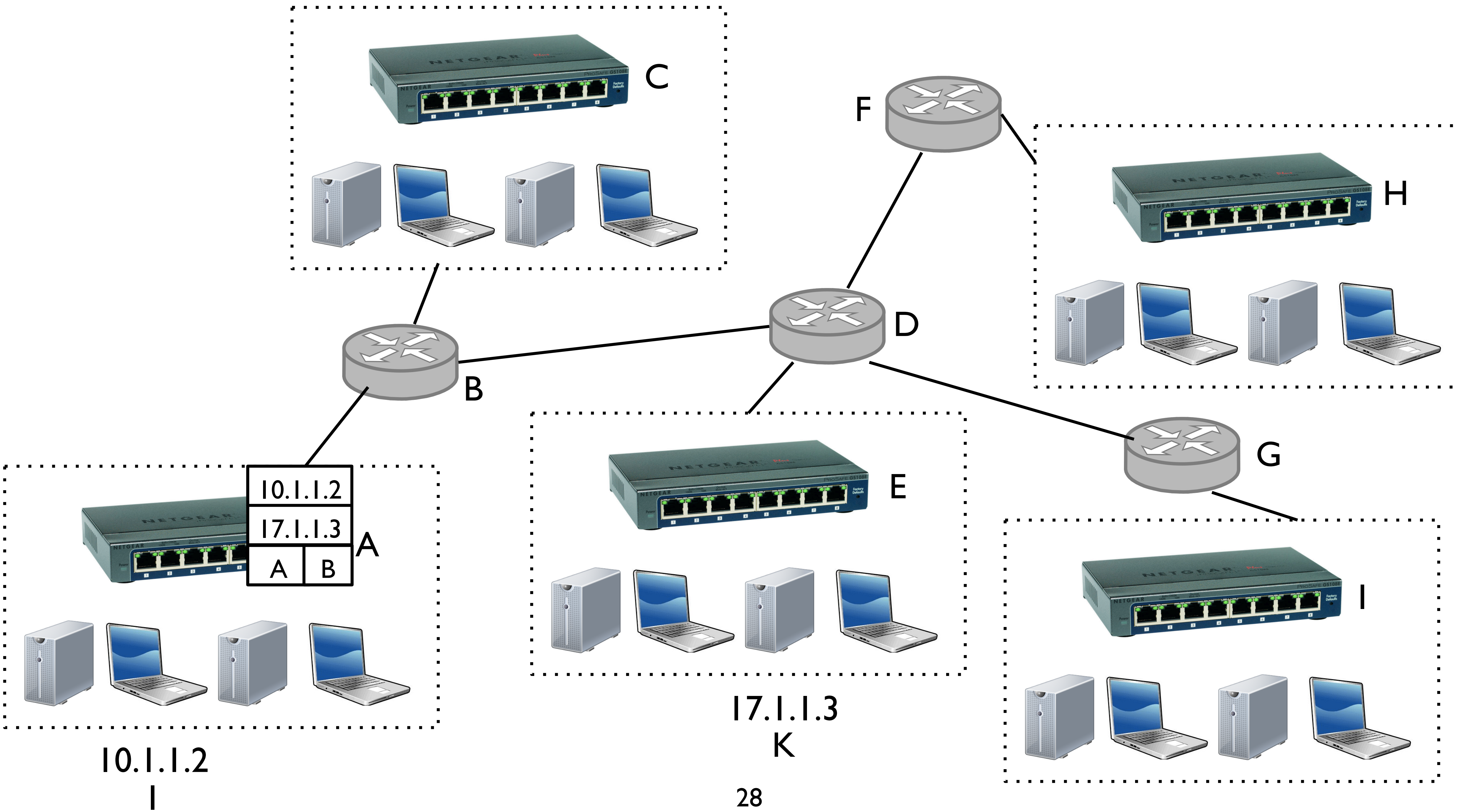


(this is what makes the Internet work)

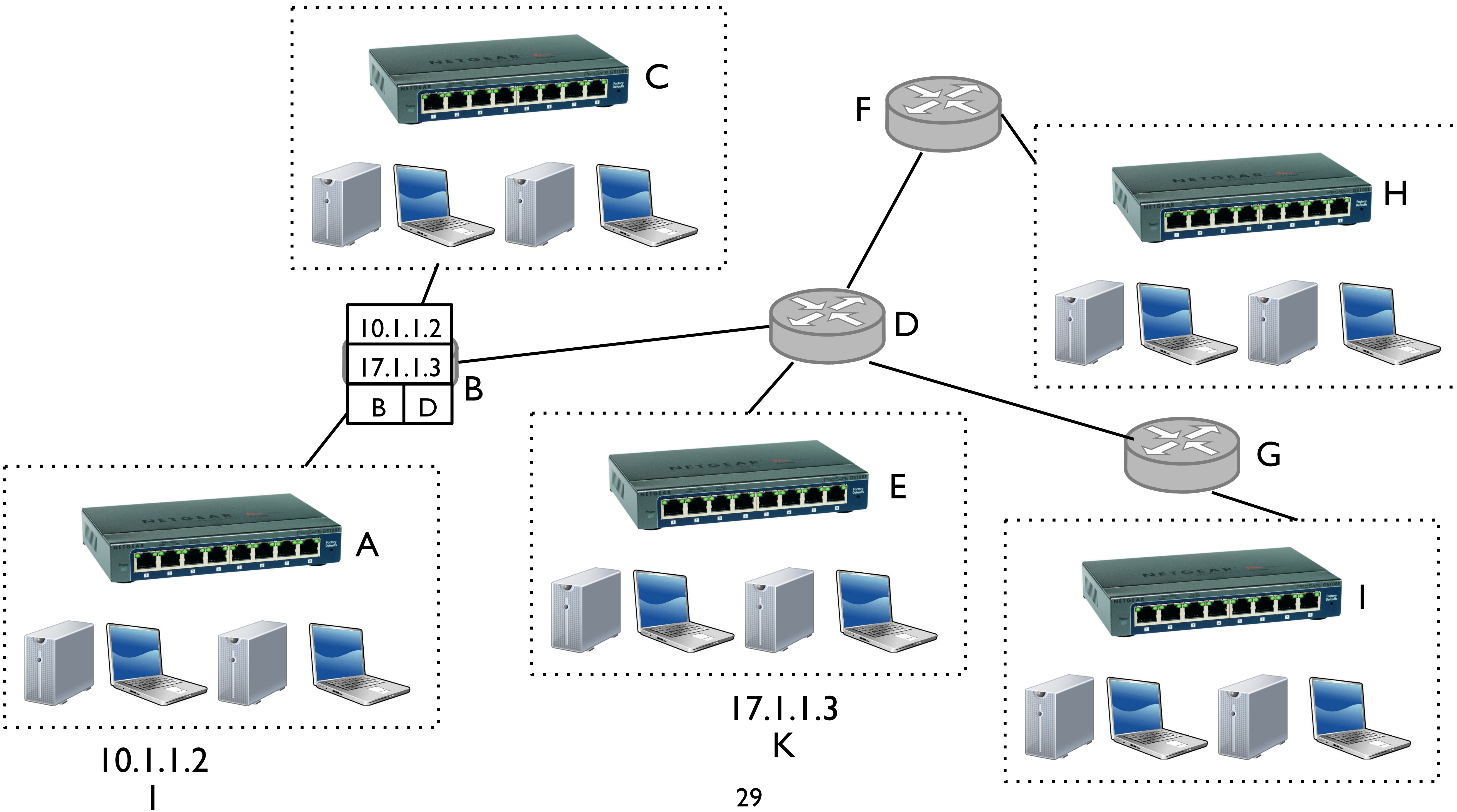
Internet routing



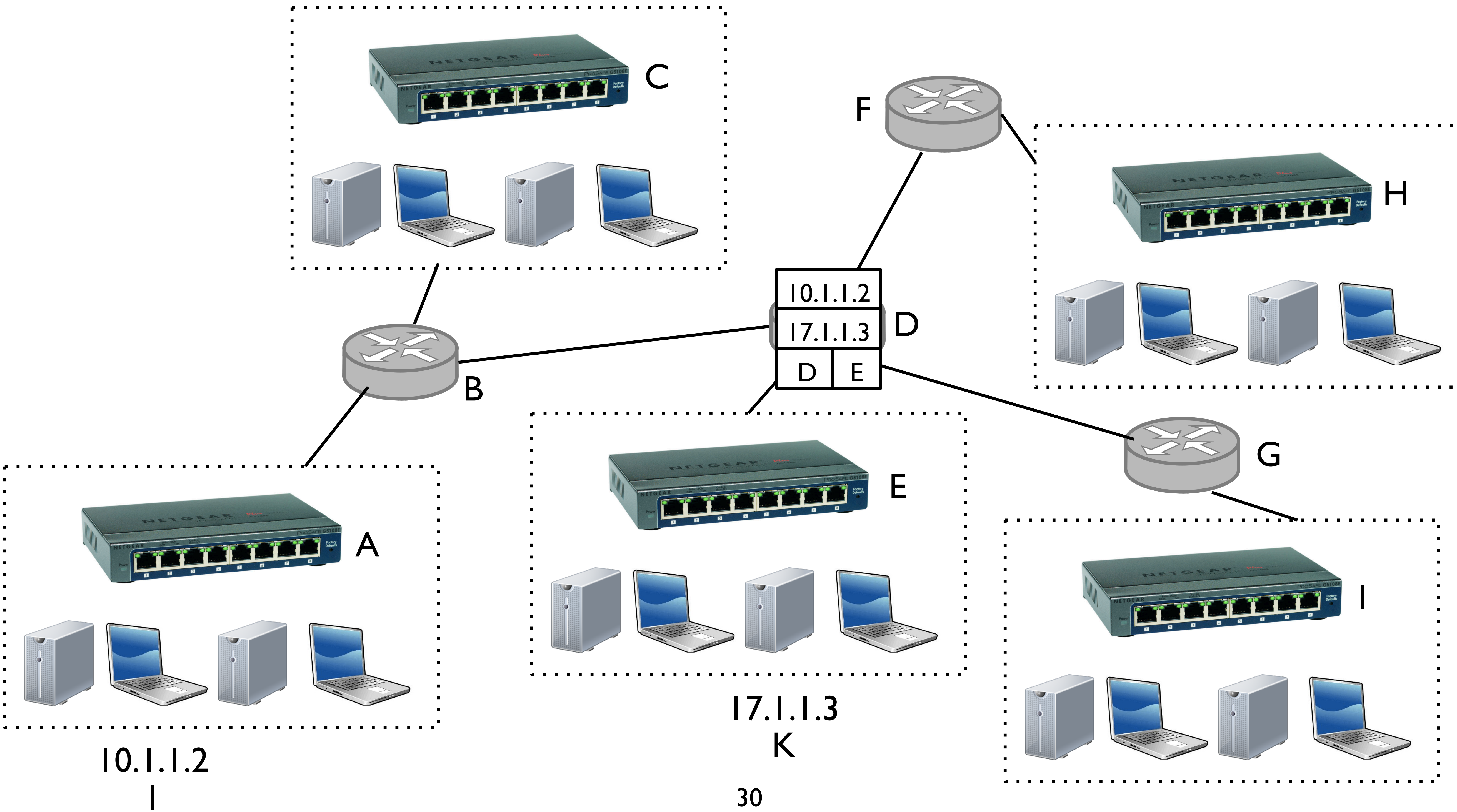
Internet routing



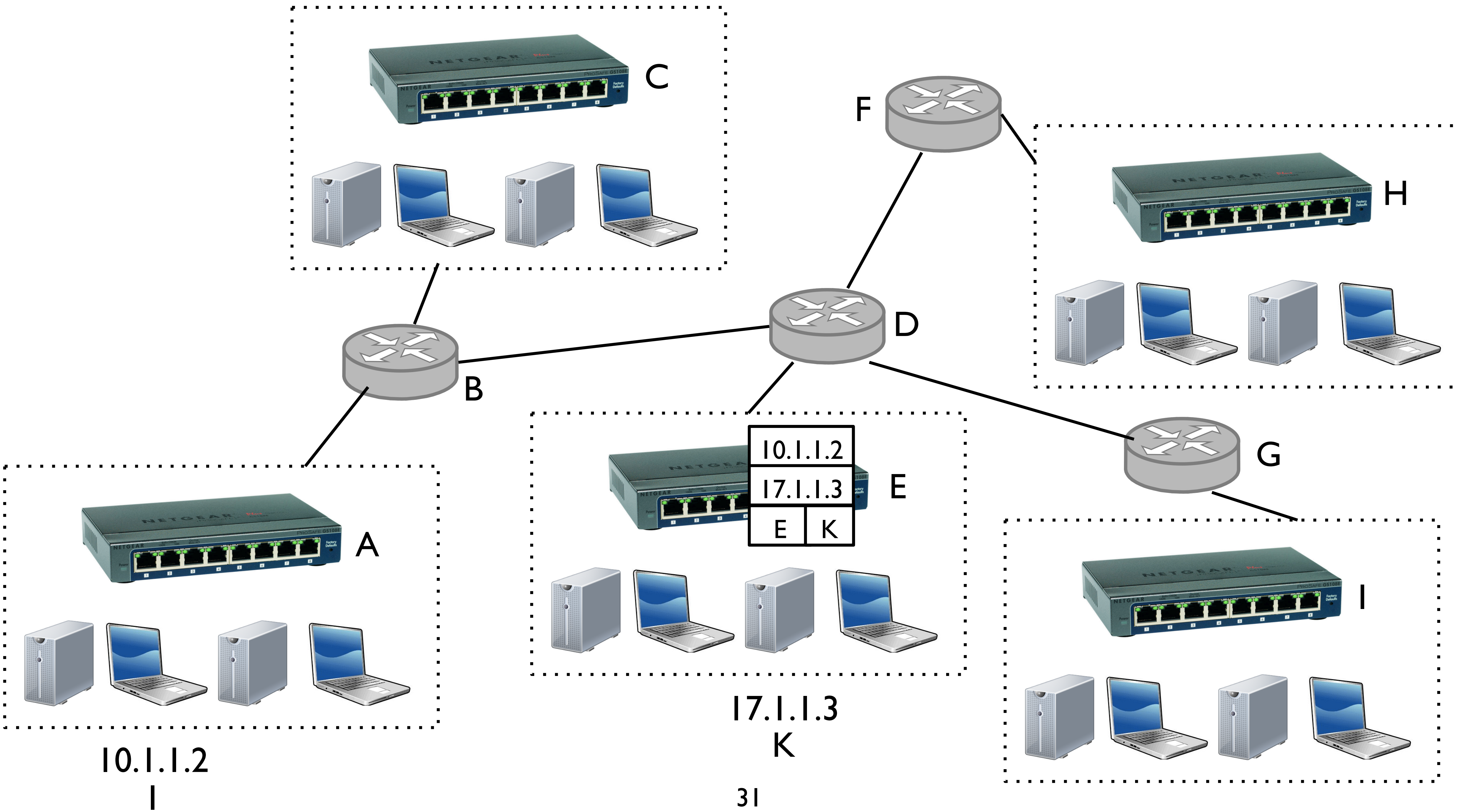
Internet routing

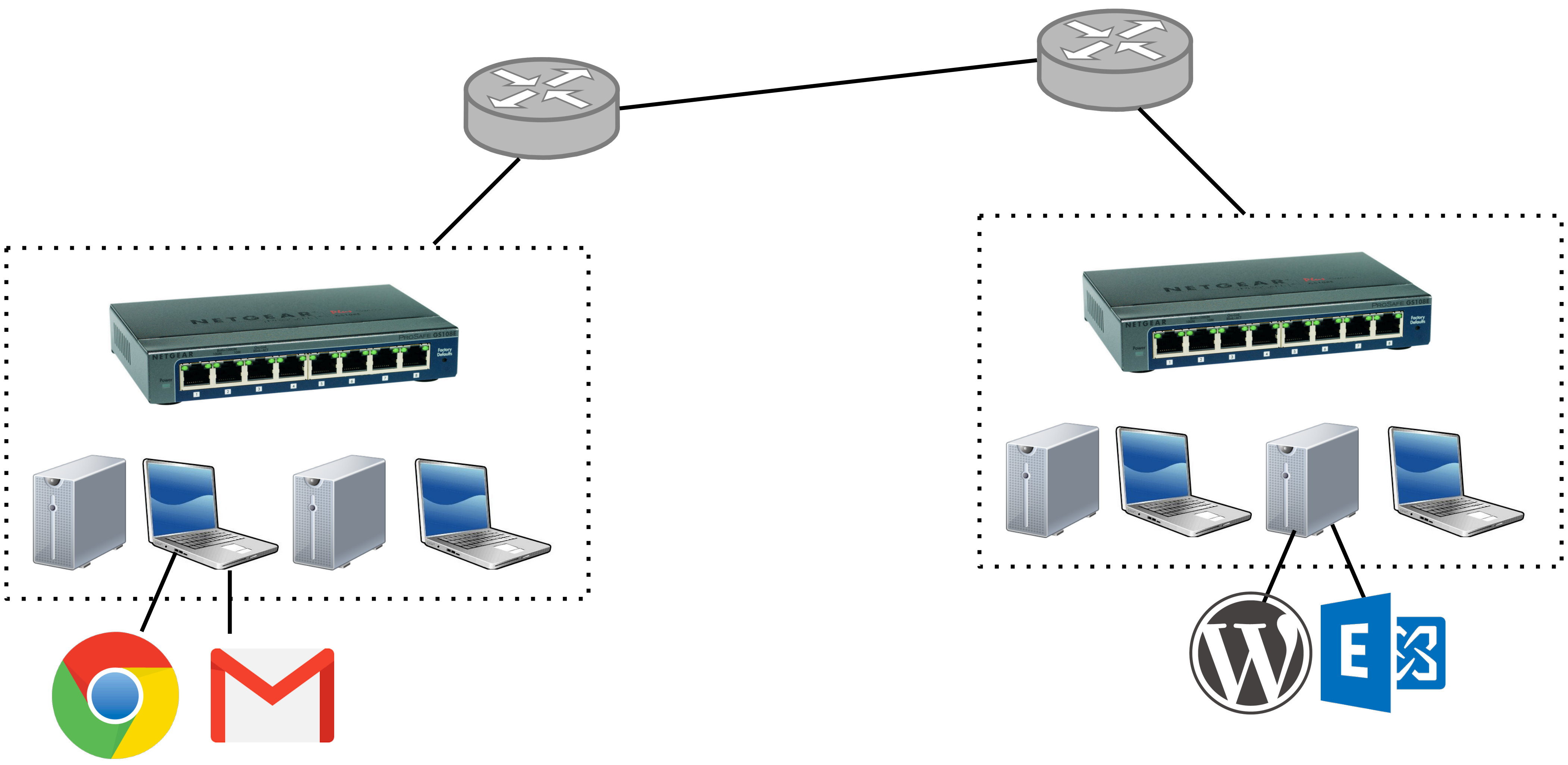


Internet routing



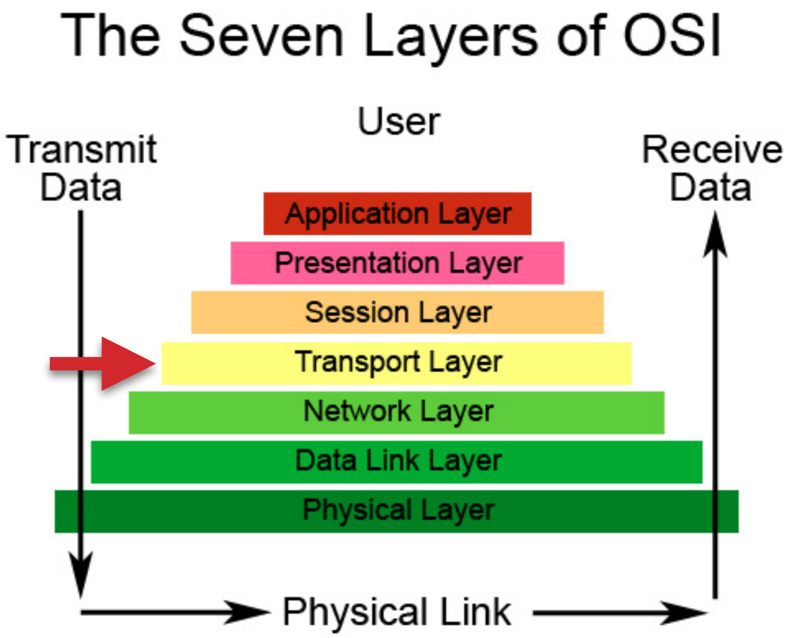
Internet routing





Transport Layer (4)

- Service:
  - Process-to-process channels
  - Demultiplexing (via ports) to different processes
  - Optional: error-free and flow-controlled delivery
- Interface: send message to specific destination (address and port)
- Protocol: implements reliability and flow control
  - Examples:
    - UDP (unreliable)
    - TCP (reliable)



Ports are for **multiplexing**

Port Number	Service
22	ssh (secure terminal)
25	Email server
80	Web server (HTTP)
443	Secure web server (HTTPS)

Destination port tells OS to which application (service) it should send incoming message

Ports are for **multiplexing**

- Internet Assigned Numbers Authority (IANA) defines list of reserved ports
- In reality, program can listen to any port in $[1, 2^{16})$
  - Ports below 1024 usually require admin privileges
  - For new apps, best to choose a high-numbered port (e.g., 9999 for UnencryptedIM)

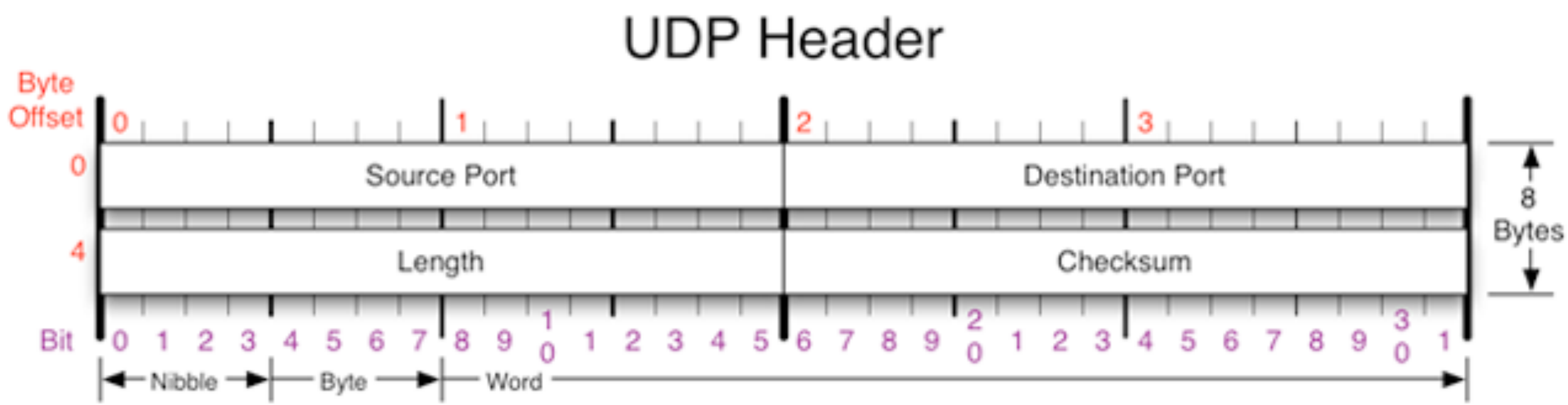
Port Number	Service
22	ssh (secure terminal)
25	Email server
80	Web server (HTTP)
443	Secure web server (HTTPS)

Destination port tells OS to which application (service) it should send incoming message

User Datagram Protocol (UDP)

- Unreliable transport
- Provides integrity protection
- Doesn't handle:
  - out of order delivery
  - lost packets
  - duplication

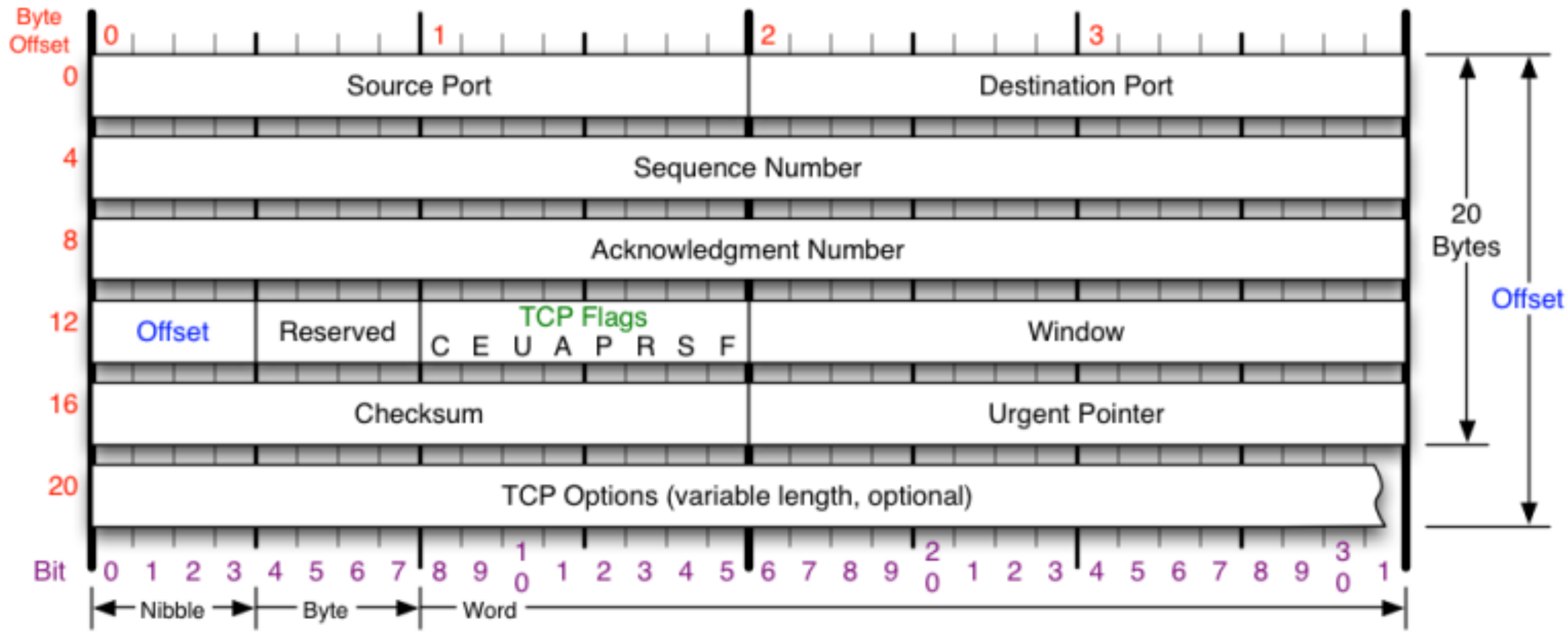
When might this be ok?



Transmission Control Protocol (TCP)

- Handles loss, duplication, and out-of-order delivery
  - relies on ACKnowledgements
- Provides flow control: prevent congestion
- Provides congestion control: handle congestion

TCP



TCP Flags

C E U A P R S F

Congestion Window
- C 0x80 Reduced (CWR)
- E 0x40 ECN Echo (ECE)
- U 0x20 Urgent
- A 0x10 Ack
- P 0x08 Push
- R 0x04 Reset
- S 0x02 Syn
- F 0x01 Fin

Congestion Notification

ECN (Explicit Congestion Notification). See RFC 3168 for full details, valid states below.

Packet State	DSB	ECN bits
Syn	0 0	1 1
Syn-Ack	0 0	0 1
Ack	0 1	0 0
No Congestion	0 1	0 0
No Congestion	1 0	0 0
Congestion	1 1	0 0
Receiver Response	1 1	0 1
Sender Response	1 1	1 1

TCP Options

0 End of Options List
1 No Operation (NOP, Pad)
2 Maximum segment size
3 Window Scale
4 Selective ACK ok
8 Timestamp

Checksum

Checksum of entire TCP segment and pseudo header (parts of IP header)

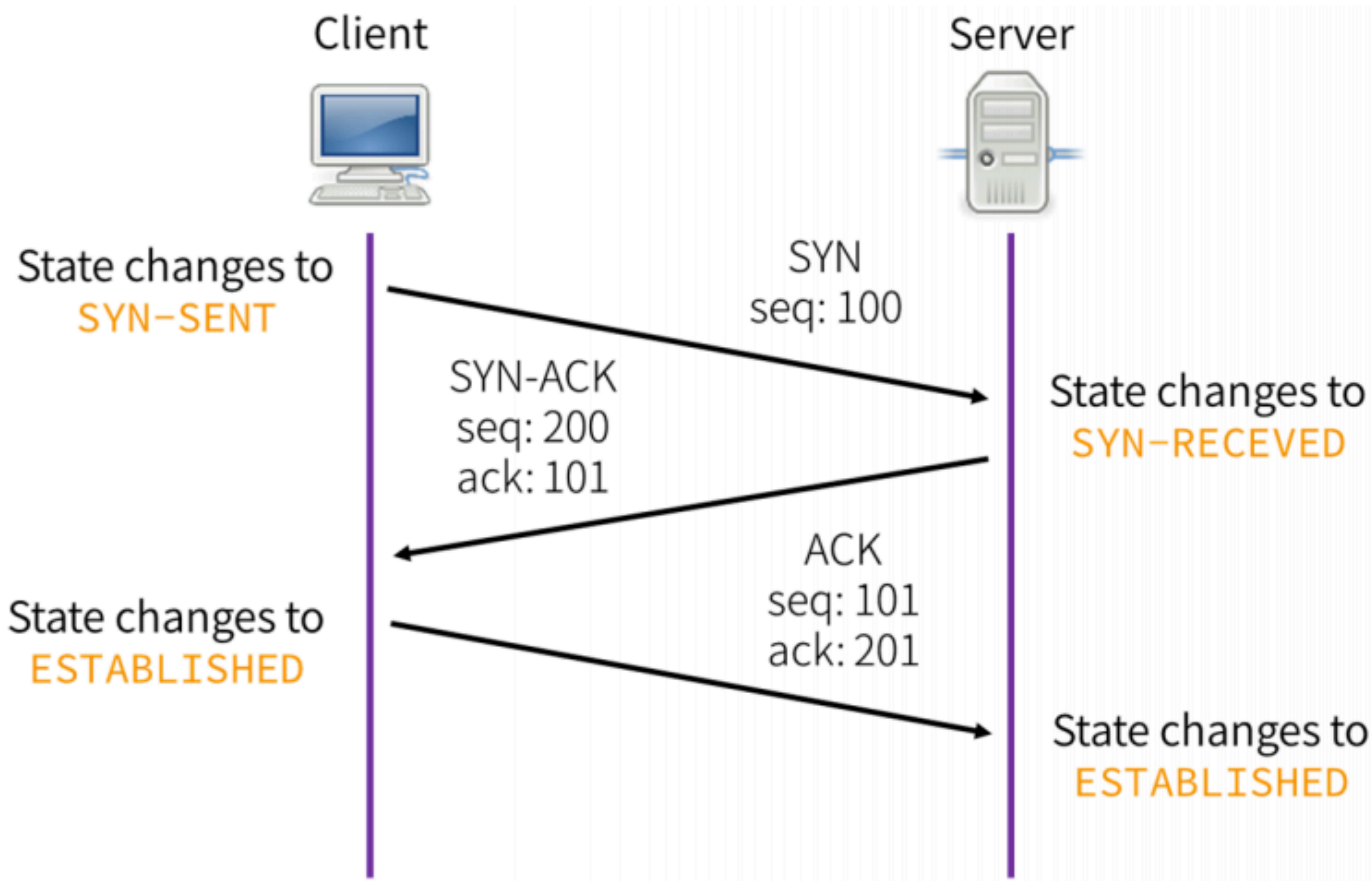
Offset

Number of 32-bit words in TCP header, minimum value of 5. Multiply by 4 to get byte count.

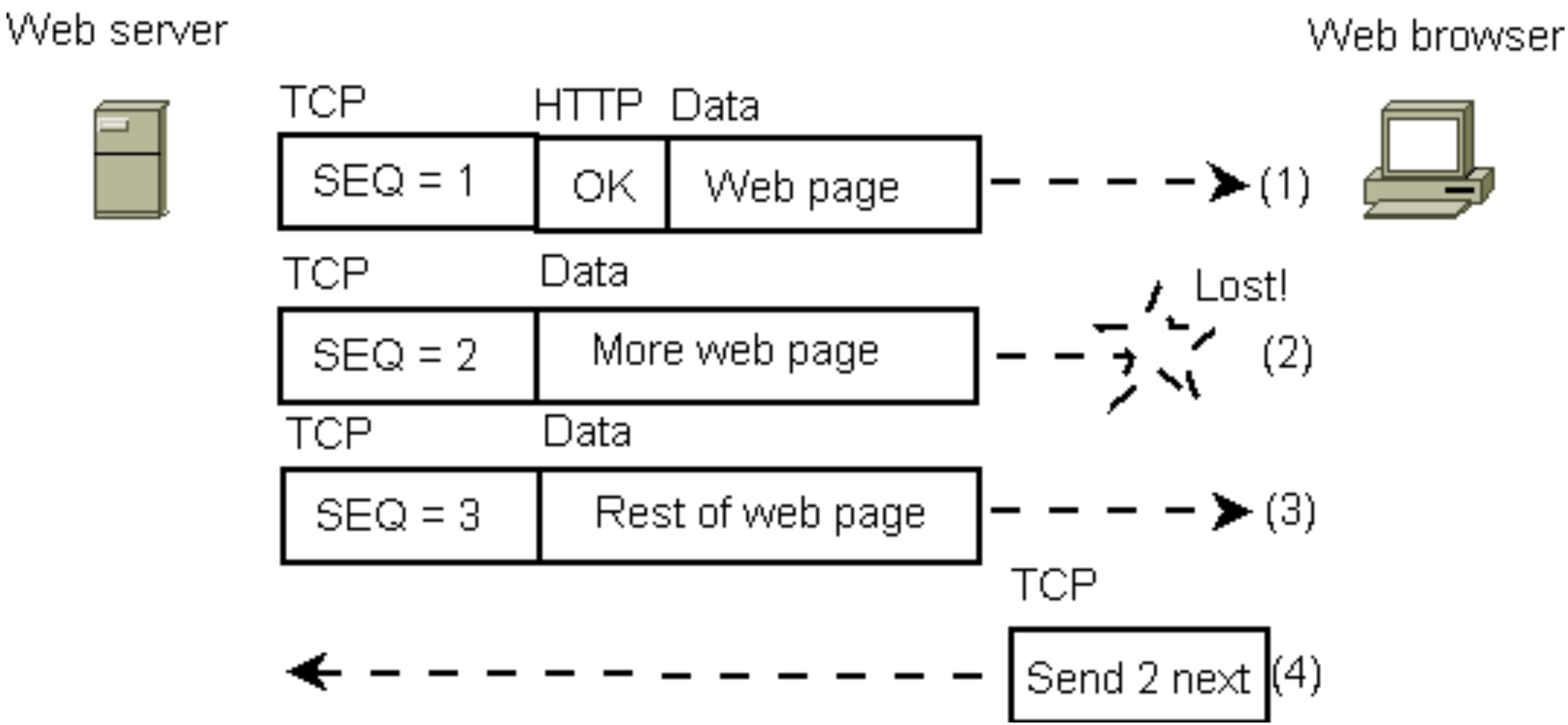
RFC 793

Please refer to RFC 793 for the complete Transmission Control Protocol (TCP) Specification.

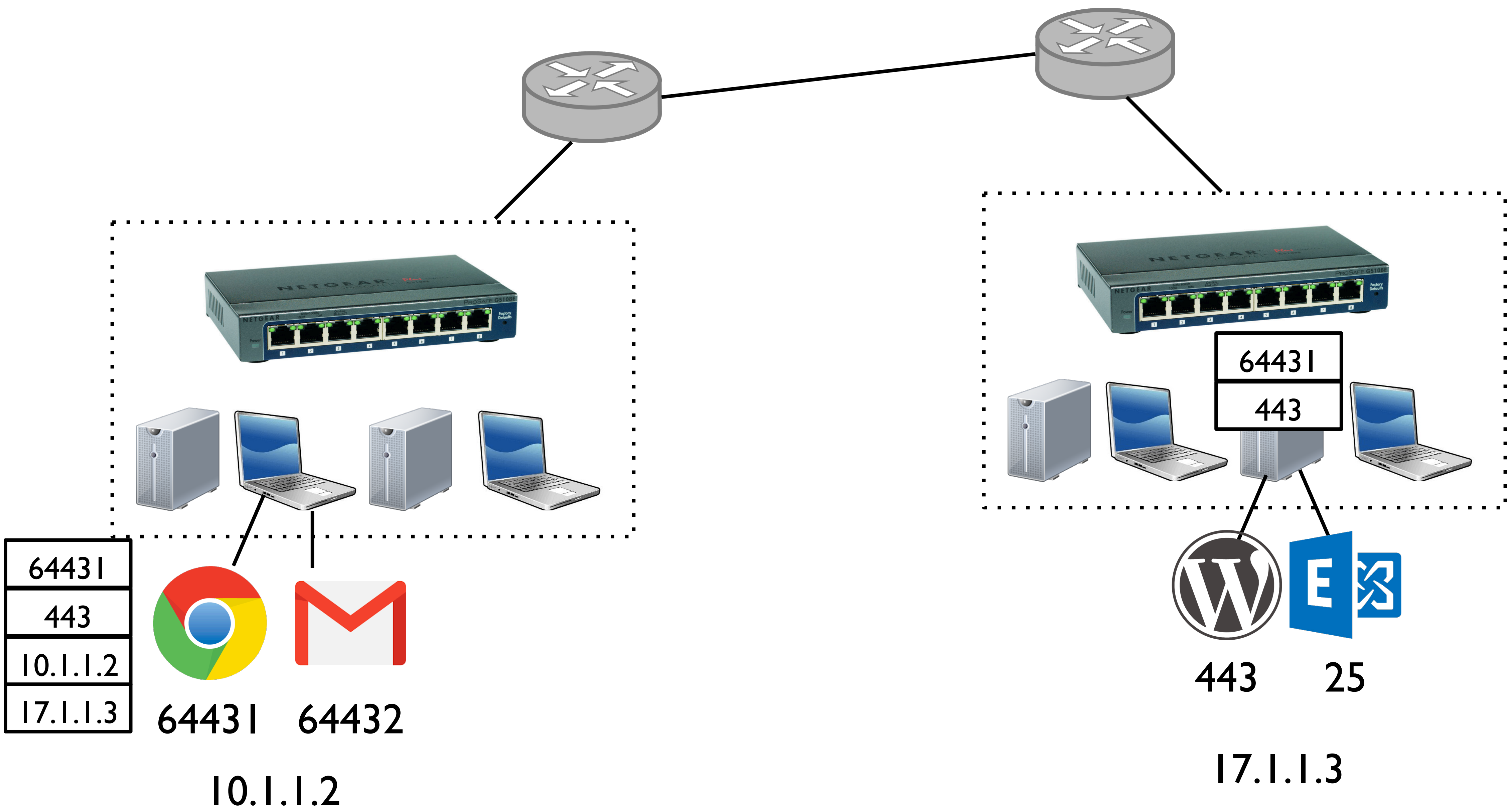
TCP Handshake



Error Handling



https://reisinge.net/notes/net/basics

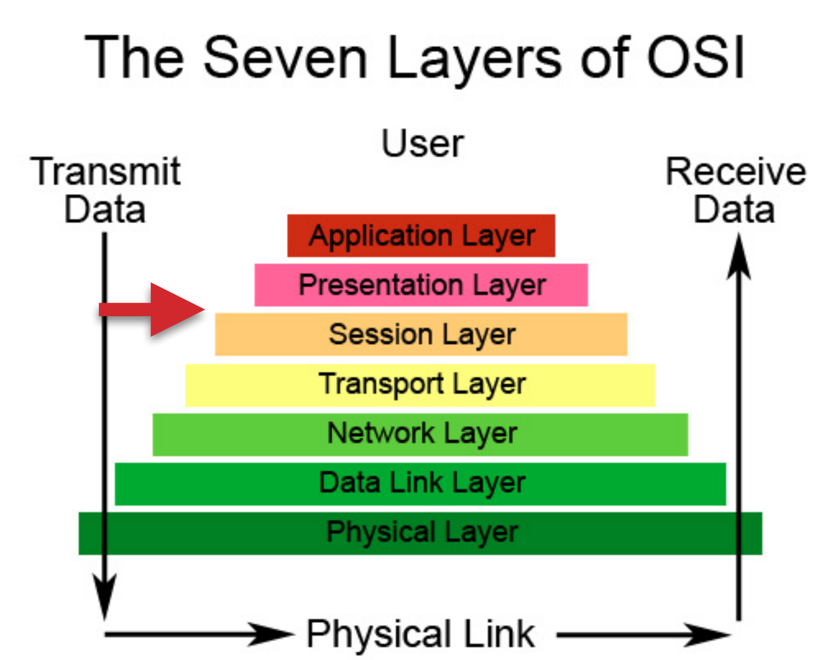


64431
443
10.1.1.2
17.1.1.3
64431
64432
10.1.1.2
64431
443
443
25
17.1.1.3

Session (5) and Presentation (6) Layers

- **Session service:**
    - Combines different transport streams for each application
    - E.g. audio and video stream in a teleconferencing application
- **Presentation service:**
    - Converts data formats between various representations to provide a standard interface for the application layer

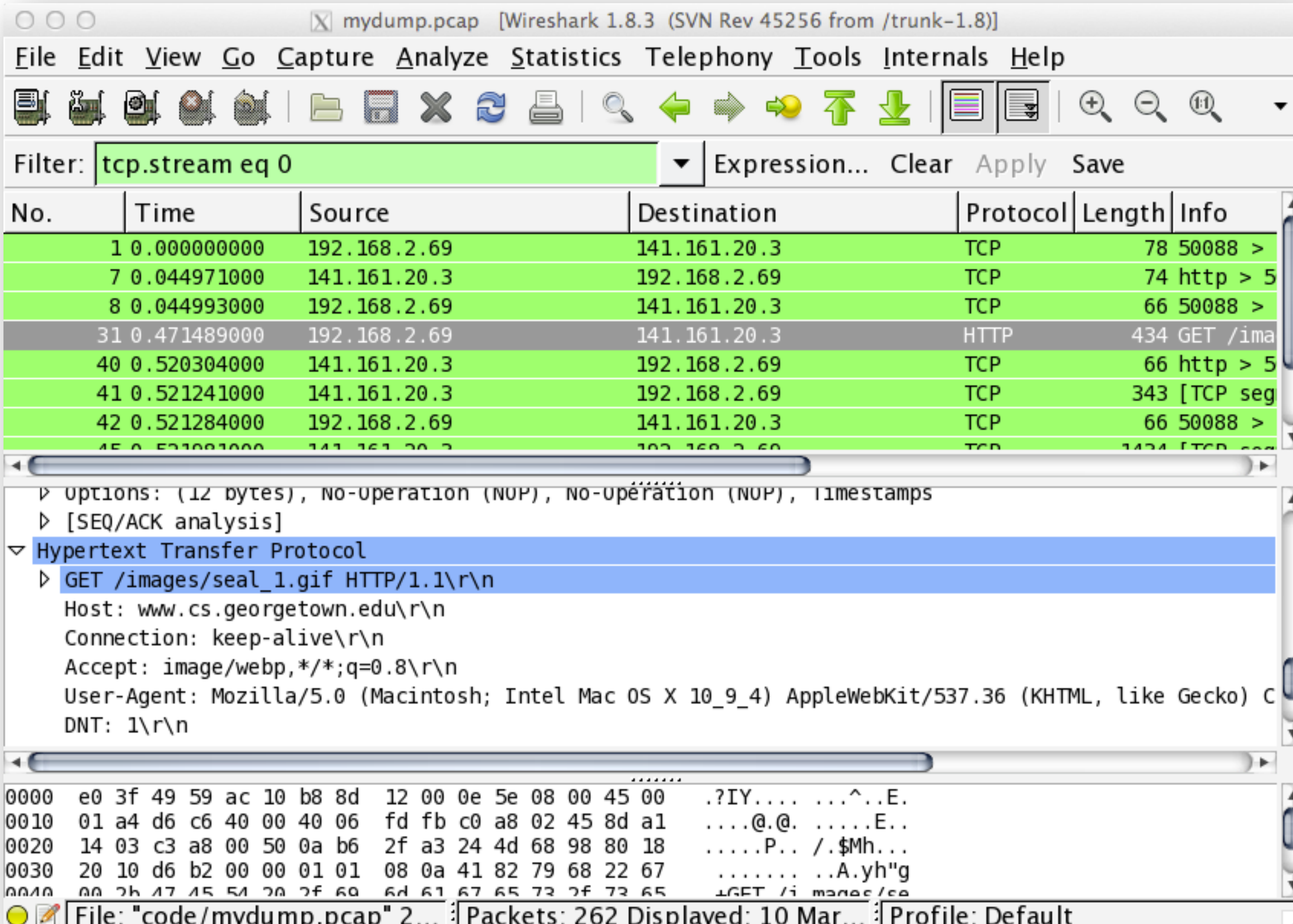
Reality:
These aren't really used.



Application Layer (7)

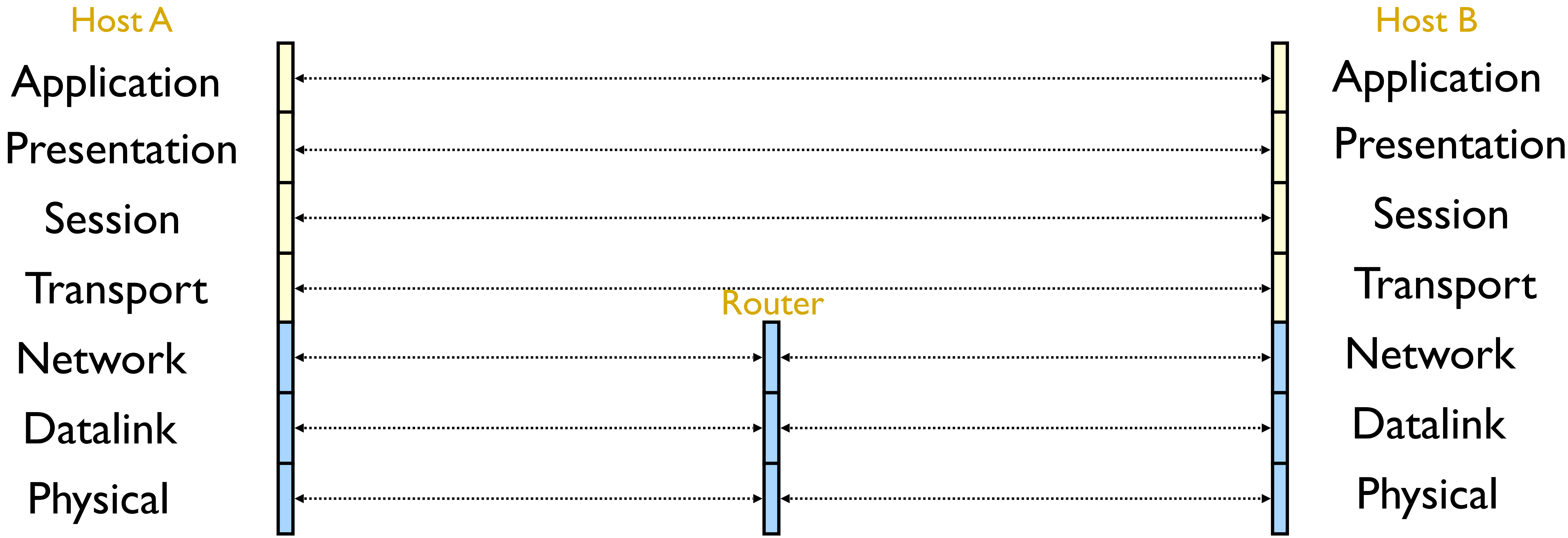
- Service: any service provided to the end user
- Interface: depends on the application
- Protocol: depends on the application
- Examples: FTP, Telnet, SSH, Email, HTTP

Demo!


mydump.pcap [Wireshark 1.8.3 (SVN Rev 45256 from /trunk-1.8)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: tcp.stream eq 0
Expression... Clear Apply Save
No. Time Source Destination Protocol Length Info
1 0.000000000 192.168.2.69 141.161.20.3 TCP 78 50088 >
7 0.044971000 141.161.20.3 192.168.2.69 TCP 74 http > 5
8 0.044993000 192.168.2.69 141.161.20.3 TCP 66 50088 >
31 0.471489000 192.168.2.69 141.161.20.3 HTTP 434 GET /ima
40 0.520304000 141.161.20.3 192.168.2.69 TCP 66 http > 5
41 0.521241000 141.161.20.3 192.168.2.69 TCP 343 [TCP seg
42 0.521284000 192.168.2.69 141.161.20.3 TCP 66 50088 >
[SEQ/ACK analysis]
Hypertext Transfer Protocol
GET /images/seal_1.gif HTTP/1.1\r\n
Host: www.cs.georgetown.edu\r\n
Connection: keep-alive\r\n
Accept: image/webp,*/*;q=0.8\r\n
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.36 (KHTML, like Gecko) C
DNT: 1\r\n
0000 e0 3f 49 59 ac 10 b8 8d 12 00 0e 5e 08 00 45 00 .?IY.... ...^..E.
0010 01 a4 d6 c6 40 00 40 06 fd fb c0 a8 02 45 8d a1 ....@.@. .....E..
0020 14 03 c3 a8 00 50 0a b6 2f a3 24 4d 68 98 80 18 .....P.. /.$Mh...
0030 20 10 d6 b2 00 00 01 01 08 0a 41 82 79 68 22 67 ....A.yh"g
File: "code/mydump.pcap" 2...
Packets: 262 Displayed: 10 Mar...
Profile: Default

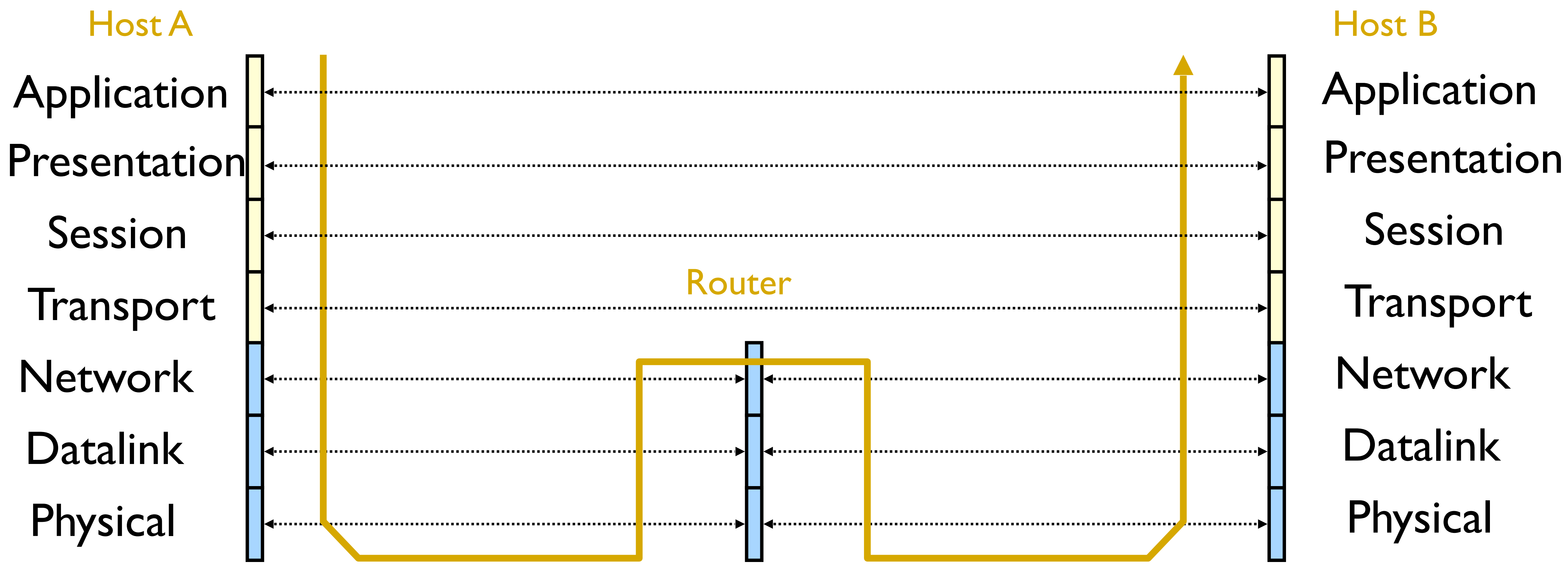
Who Does What?

- Lower three layers are implemented everywhere
- Next four layers are implemented only at hosts



Physical Communication

- Communication goes down to physical network, then to peer, then up to relevant layer



What about security?

- Where is confidentiality, integrity, and authenticity?
- No relevant "security" fields in IP, TCP, or UDP headers.
- Why not?

Network Programming

- The operating system provides an *interface* for sending/receiving network packets
- A ***socket*** is a descriptor for network communication
- As a client, you ***connect*** your socket to a remote host, and read/write to that socket as you would a file
- As a server, you ***listen*** and ***accept*** incoming connections, and read/write to that socket as you would a file

Internet communication *via sockets*

Src=HostA,SrcPort=1234
Dest=HostB,DestPort=1025
Content="Hello world"

HostA

HostB

running on HostA

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# connect to HostB on port 1025.
s.connect( (”HostB",1025) )
# Let the Internet worry about how my messages get there
s.send('hello world’)
s.close()
```

Internet communication


```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((‘’, 1025))  # this socket is bound to my port 1025
s.listen(1)         # specify the “backlog” for this socket
conn, addr = s.accept()  # wait and accept the connection
data = conn.recv(1024)   # read the content of the message
print data;
conn.close()
```

running on HostB

Network Programming

- The operating system provides an *interface* for sending/ receiving network packets
- A ***socket*** is a descriptor for network communication
- As a client, you ***connect*** your socket to a remote host, and read/write to that socket as you would a file
- As a server, you ***listen*** and ***accept*** incoming connections, and read/write to that socket as you would a file
- read()/recv() is a blocking operation; to wait for input from multiple sources, use ***select***

Select example

```python
import socket,select

listen_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
listen_socket.bind(('', 9998))
listen_socket.listen(10)
client_sockets = []                     # an empty list

while True:
    read_list = [listen_socket] + client_sockets
    (ready_read,_,_) = select.select(read_list,[],[])

    for sock in ready_read:
        if sock is listen_socket:
            new_conn, addr = sock.accept()  # accept the connection
            client_sockets.append(new_conn)
        else:
            data = sock.recv(1024)          # read up to 1K of data
            if data != "":                  # the connection is open
                sock.send("Go away.\n")     #   I'm not very nice
            else:                           # the connection is closed
                client_sockets.remove(sock)
                sock.close()
```