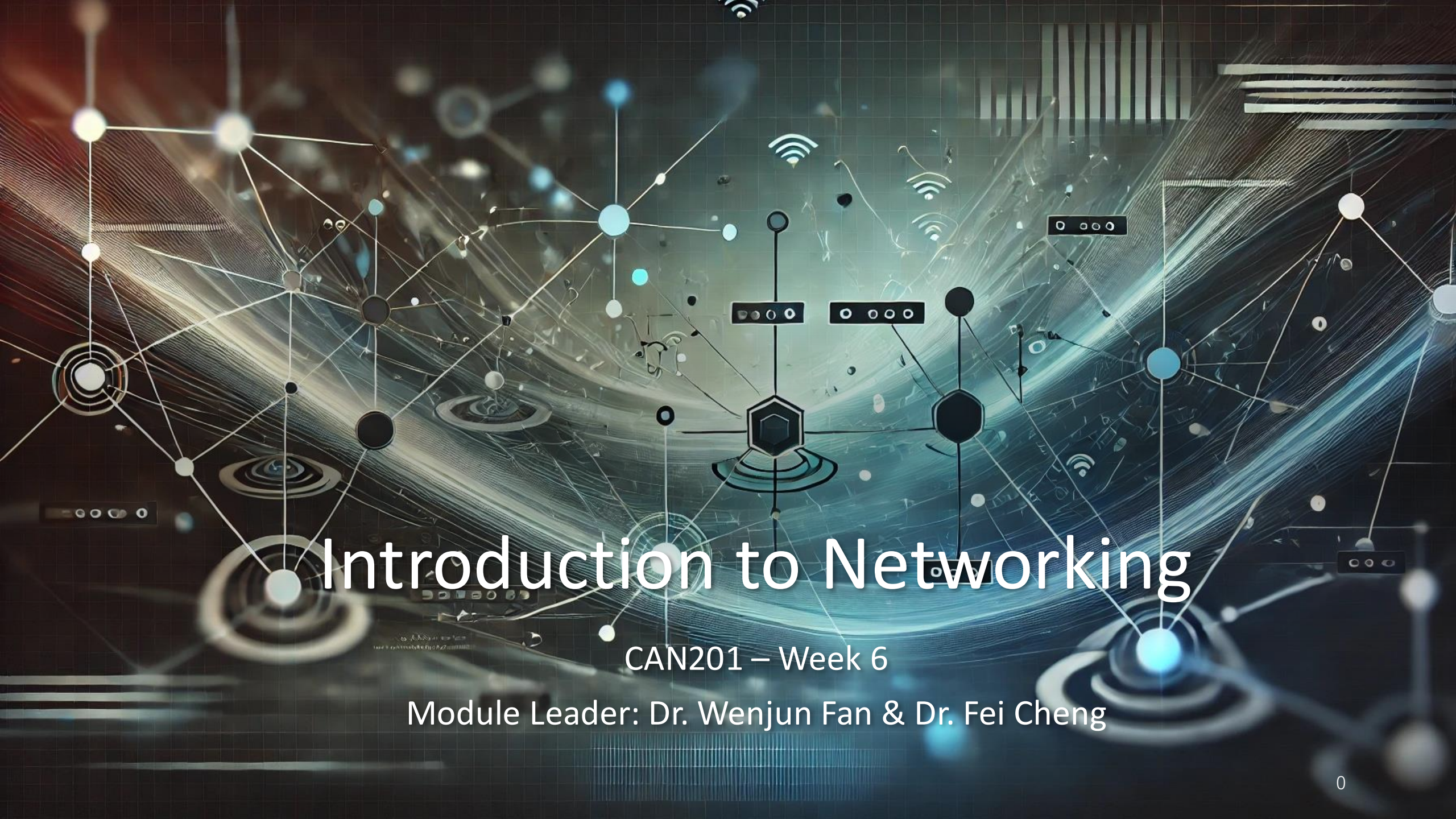


Introduction to Networking

CAN201 – Week 6

Module Leader: Dr. Wenjun Fan & Dr. Fei Cheng

Lecture 6 – Transport Layer (3)

- **Roadmap**
  1. TCP congestion control

Approaches to Congestion Control

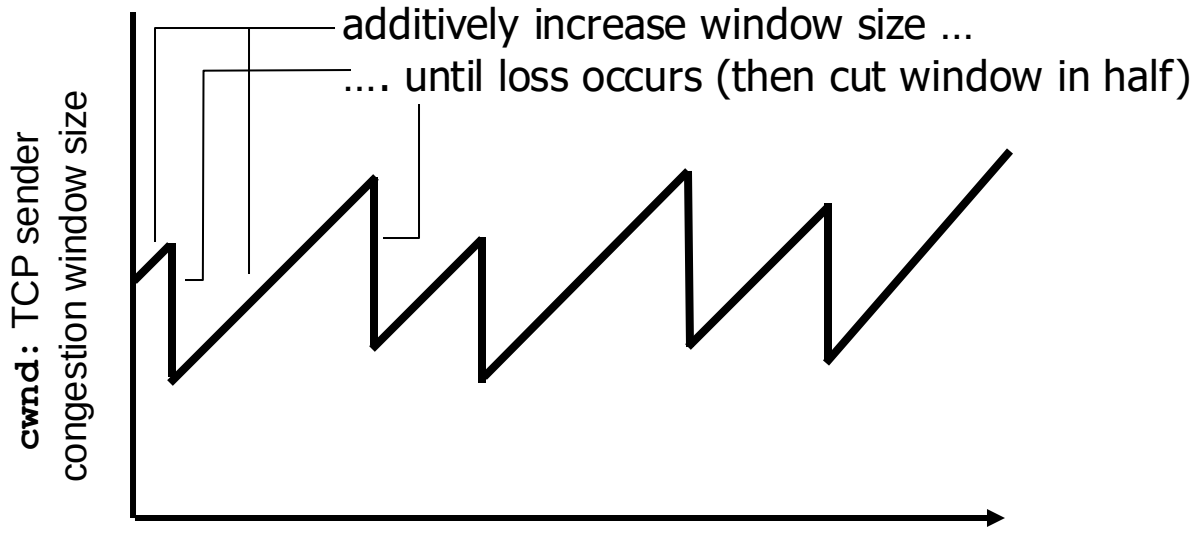
- **End-to-end congestion control**
  - Network layer does not provide support for congestion control
  - Trans layer has to infer from network behavior
  - TCP will control the size of window
- **Network-assisted congestion control**
  - Routers provide feedback to sender and/or receiver (a single one bit)

TCP Congestion Control

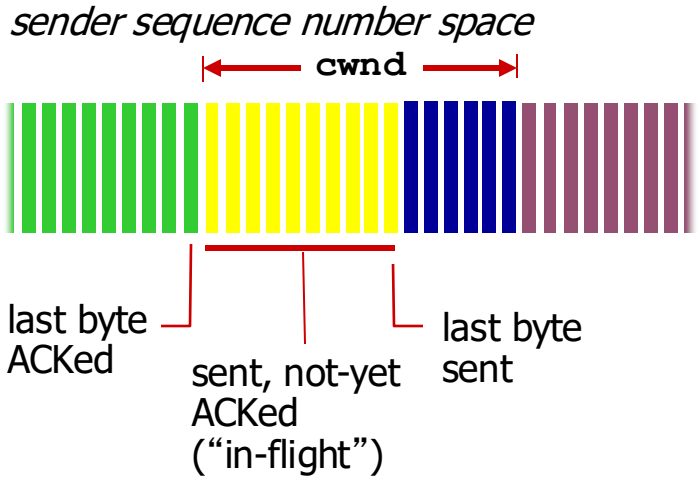
- **Strategy: additive increase multiplicative decrease (AIMD)**
- **Approach: sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs**
  - *Additive increase:* increase **cwnd (**congestion window**)** by 1 MSS（Maximum Segment Size）every RTT until loss detected
  - *Multiplicative decrease*: cut **cwnd** in half after loss

TCP Congestion Control

AIMD saw tooth behavior: probing for bandwidth



TCP Congestion Control: details



- Sender limits transmission:

```
LastByteSent - LastByteAcked <= min(cwnd, rwnd)
```

- cwnd is dynamic, function of perceived network congestion

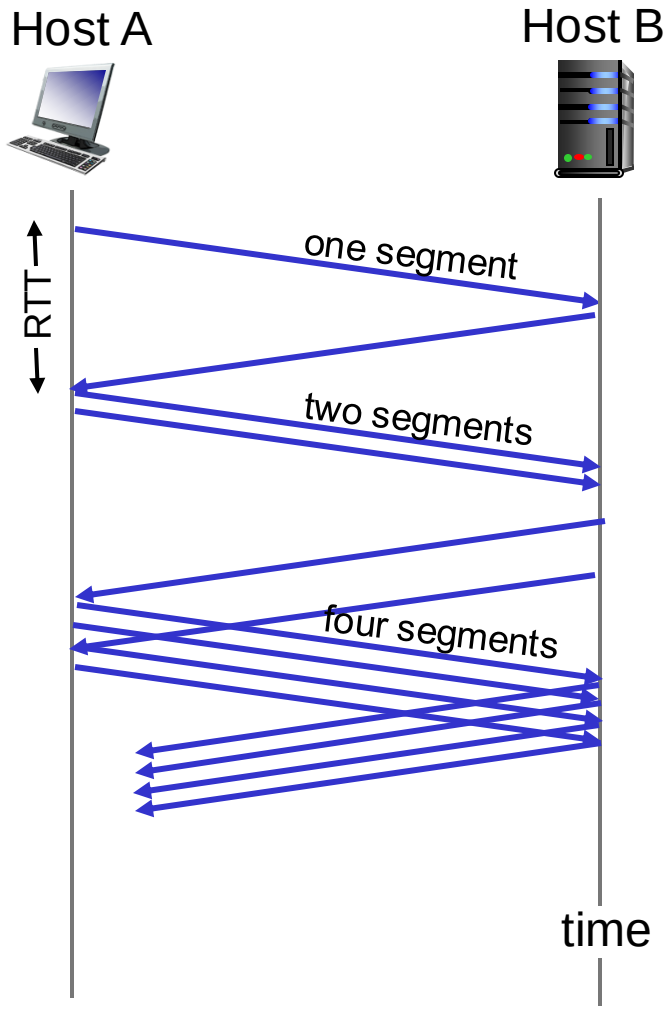
TCP sending rate:

- Roughly: send cwnd bytes, wait RTT for ACKS, then send more bytes

$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

TCP Slow Start

- **When connection begins, increase rate exponentially until first loss event:**
  - Initially **`cwnd`** = 1 MSS* (maximum segment size)
  - Double **`cwnd`** every RTT
  - Done by incrementing **`cwnd`** for every ACK received
- ***Summary:*** **initial rate is slow but ramps up exponentially fast**



*MSS is a dynamic value, default = 536(576-40), for Ethernet max value = 1460(1500-40)

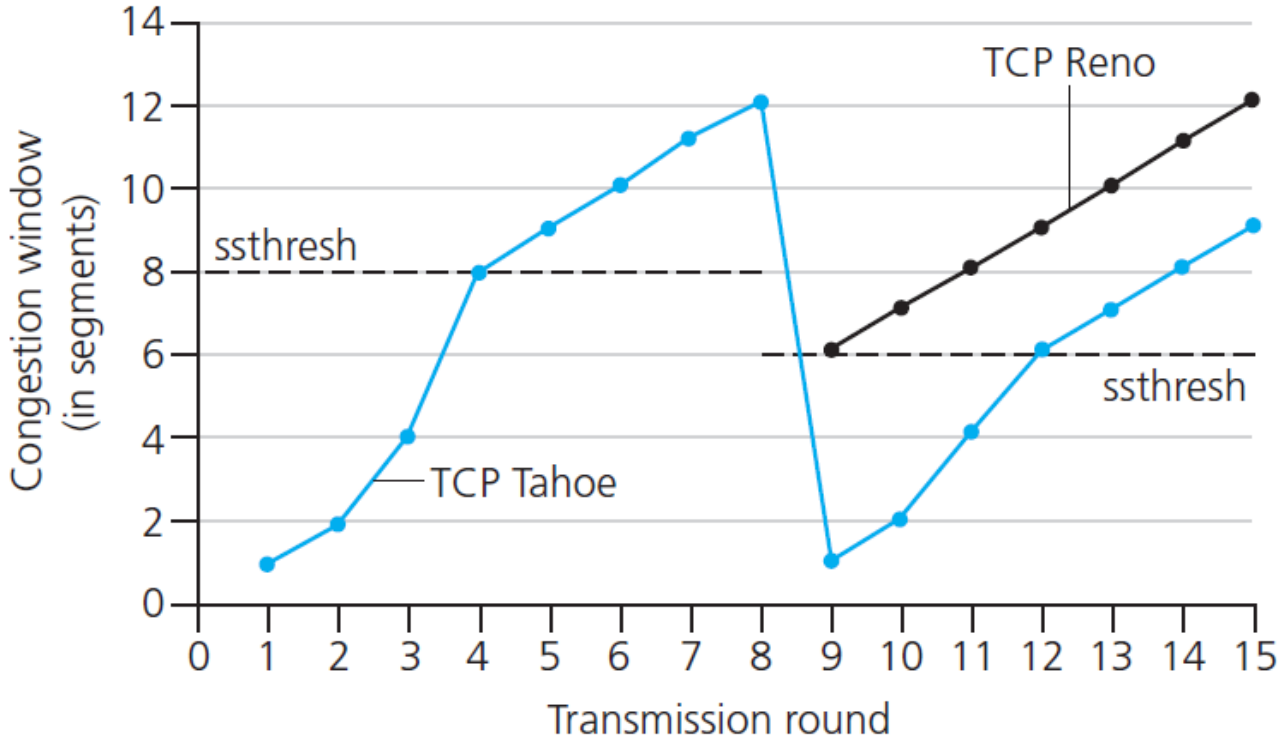
TCP: detecting, reacting to loss

- **Loss indicated by timeout:**
  - cwnd set to 1 MSS;
  - window then grows exponentially (as in slow start) to threshold, then grows linearly
- **Loss indicated by 3 duplicate ACKs: TCP RENO**
  - Dup ACKs indicate network capable of delivering some segments
  - cwnd is cut in half window then grows linearly
- **TCP Tahoe always sets cwnd to 1 (timeout or 3 duplicate acks)**

TCP: from slow start to Congestion Avoidance

Q: When should the exponential increase switch to linear?

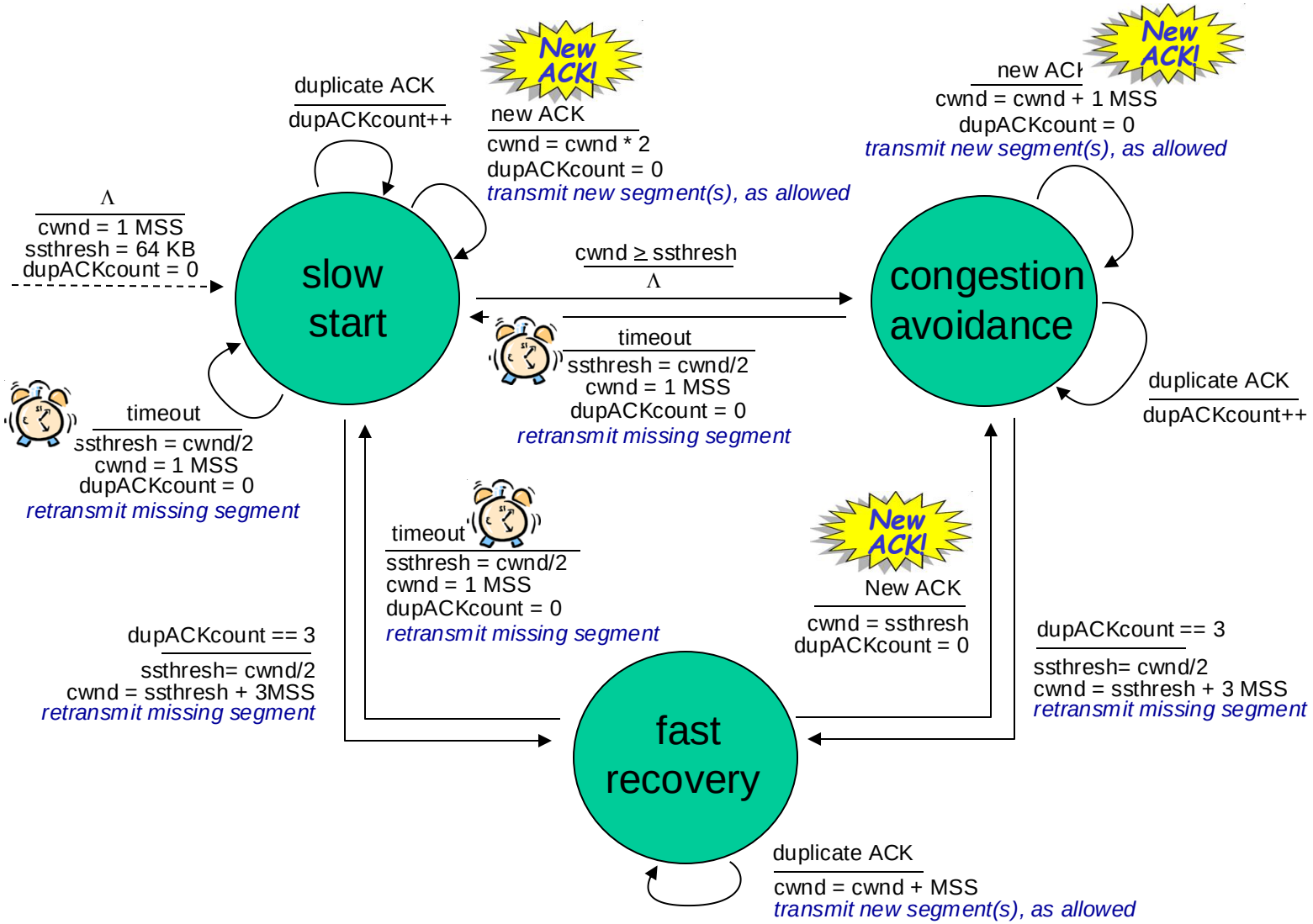
A: When `cwnd` gets to 1/2 of its value before timeout.



Implementation:

- variable **`ssthresh`**
- on loss event, **`ssthresh`** is set to 1/2 of **`cwnd`** just before loss event

Summary: TCP Congestion Control



More TCP Congestion Control Methods

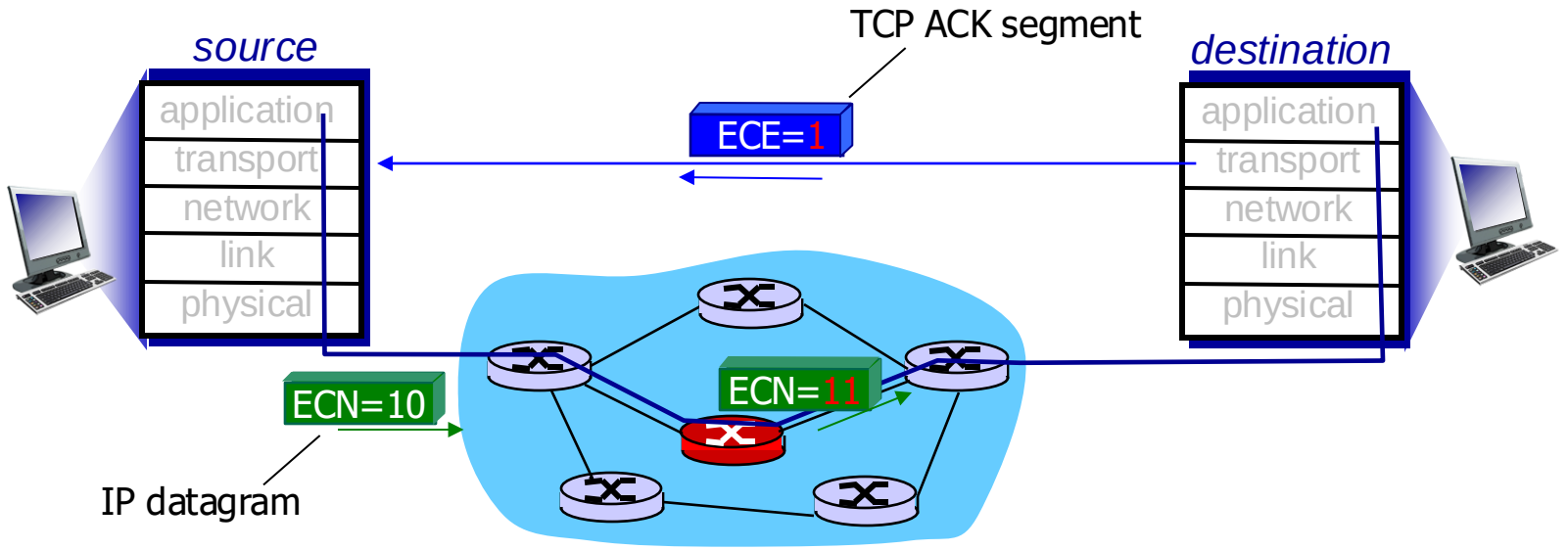
Variant	Feedback	Required changes	Benefits	Fairness
(New) Reno	Loss	—	—	Delay
Vegas	Delay	Sender	Less loss	Proportional
High Speed	Loss	Sender	High bandwidth	
BIC	Loss	Sender	High bandwidth	
CUBIC	Loss	Sender	High bandwidth	
C2TCP[9][10]	Loss/Delay	Sender	Ultra–low latency and high bandwidth	
NATCP[11]	Multi–bit signal	Sender	Near Optimal Performance	
Elastic–TCP	Loss and Delay	Sender	High bandwidth/short & long–distance	
Agile–TCP	Loss	Sender	High bandwidth/short–distance	
H–TCP	Loss	Sender	High bandwidth	
FAST	Delay	Sender	High bandwidth	Proportional
Compound TCP	Loss/Delay	Sender	High bandwidth	Proportional
Westwood	Loss/Delay	Sender	L	
Jersey	Loss/Delay	Sender	L	
BBR[12]	Delay	Sender	BLVC, Bufferbloat	
CLAMP	Multi–bit signal	Receiver, Router	V	Max–min
TFRC	Loss	Sender, Receiver	No Retransmission	Minimum delay
XCP	Multi–bit signal	Sender, Receiver, Router	BLFC	Max–min
VCP	2–bit signal	Sender, Receiver, Router	BLF	Proportional
MaxNet	Multi–bit signal	Sender, Receiver, Router	BLFSC	Max–min
JetMax	Multi–bit signal	Sender, Receiver, Router	High bandwidth	Max–min
RED	Loss	Router	Reduced delay	
ECN	Single–bit signal	Sender, Receiver, Router	Reduced loss	

https://en.wikipedia.org/wiki/TCP_congestion_control

Explicit Congestion Notification (ECN)

Network-assisted congestion control:

- Two bits in IP header (ToS field) marked *by network router* to indicate congestion
- Congestion indication carried to receiving host
- Receiver (seeing congestion indication in IP datagram)) sets ECE bit on receiver-to-sender ACK segment to notify sender of congestion



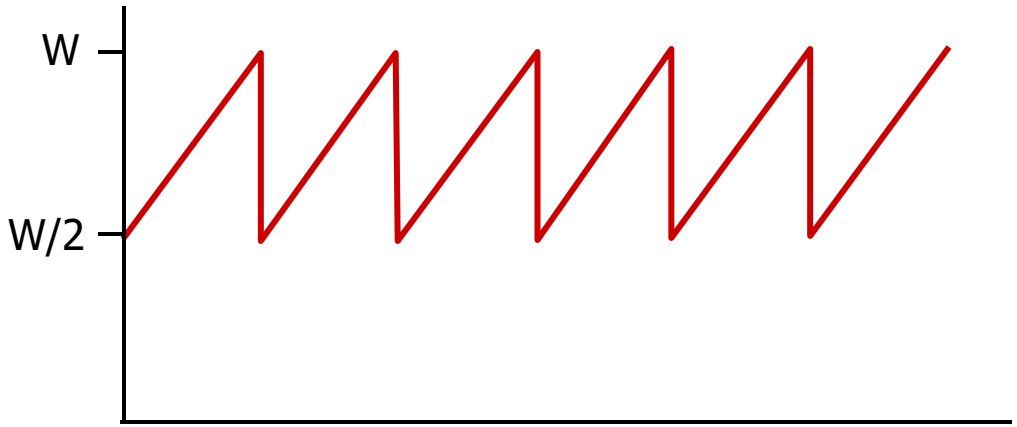
ECN:
00 Not-ECT*
01 ECT
10 ECT
11 CE**

*ECN-Capable Transport
** CE: Congestion Experienced

TCP throughput

- **avg. TCP throughput as function of window size, RTT?**
  - ignore slow start, assume always data to send
- **W: window size (measured in bytes) where loss occurs**
  - avg. window size (# in-flight bytes) is ¾ W
  - avg. thruput is 3/4W per RTT

$$\text{avg TCP thruput} = \frac{3}{4}\frac{W}{RTT} \text{ bytes/sec}$$



Summary

- **Principles behind transport layer services:**
  - Multiplexing, demultiplexing
  - Reliable data transfer
  - Flow control
  - Congestion control
- **Instantiation, implementation in the Internet**
  - UDP
  - TCP

Lecture 6 – Network Layer (1)

- **Roadmap**
  1. Overview of Network layer
  2. Router
  3. Internet Protocol

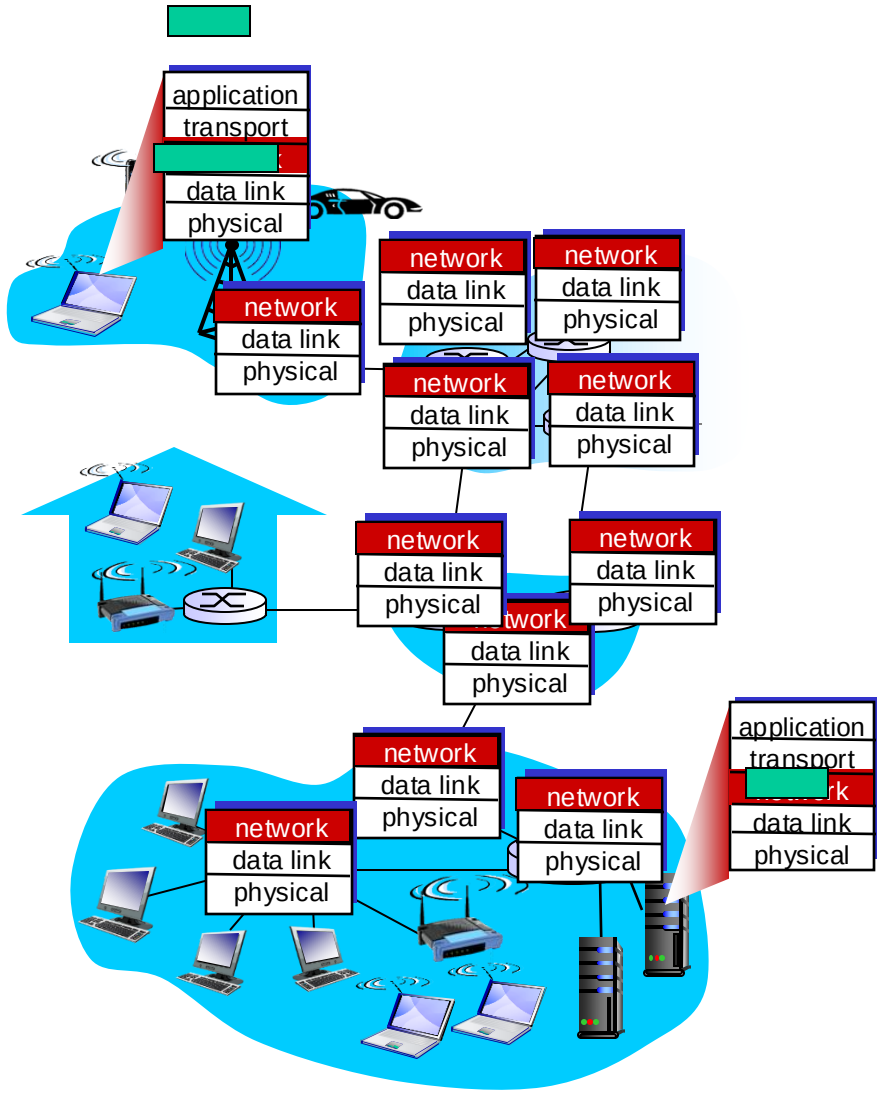
Network layer

Goal

- **Understand principles behind network layer services, focusing on data plane:**
  - Network layer service models
  - Forwarding versus routing
  - How a router works
  - Generalized forwarding
- **Instantiation, implementation in the Internet**

Network layer

- Transport segment from sending to receiving host
- On sending side encapsulates **segments** into **datagrams**
- On receiving side, delivers **segments** to transport layer
- Network layer protocols in ***every*** **host, router**
- Router examines header fields in all IP datagrams passing through it



Two key network-layer functions

Network-layer functions:

- **Forwarding: move packets from router's input to appropriate router output**
- **Routing: determine route taken by packets from source to destination**
  - Routing algorithms

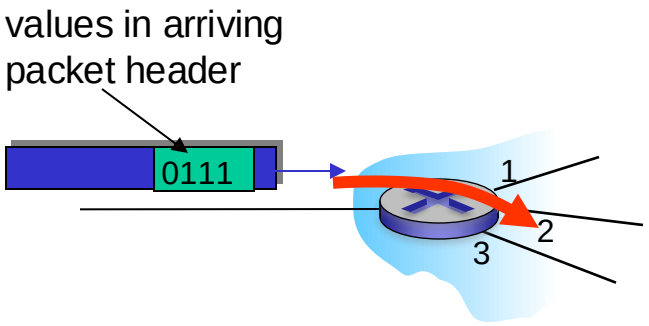
Analogy: taking a trip

- forwarding: process of getting through single interchange
- routing: process of planning trip from source to destination

Network layer: data plane, control plane

Data plane

- Local, per-router function
- Determines how datagram arriving on router input port is forwarded to router output port
- Forwarding function

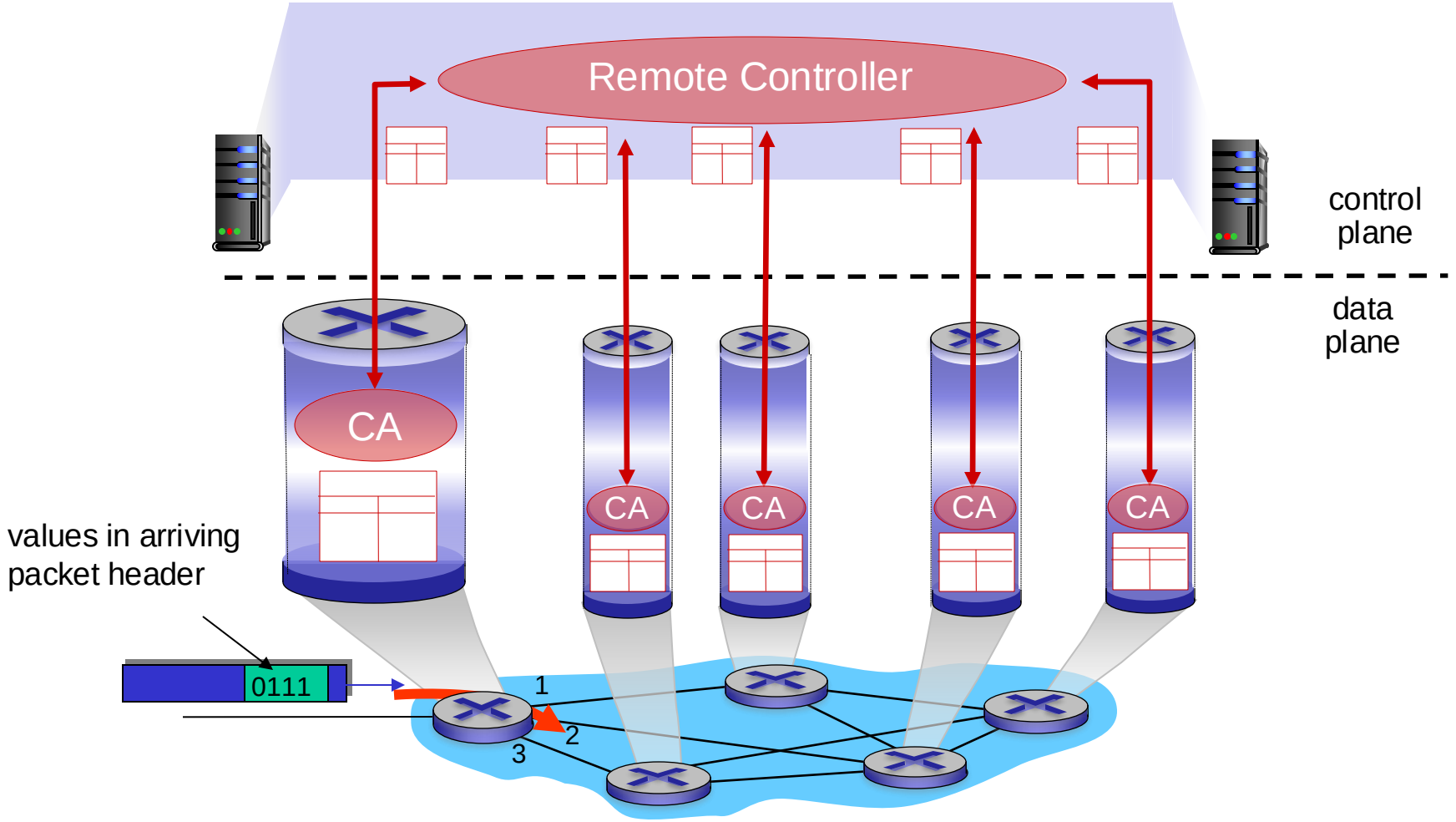


Control plane

- Network-wide logic
- Determines how datagram is routed among routers along end-end path from source host to destination host
- Two control-plane approaches:
  - *traditional routing algorithms:* implemented in routers
  - *software-defined networking (SDN)*: implemented in (remote) servers

Per-router control plane

A distinct (typically remote) controller interacts with local control agents (CAs)



Network service model

Q: **What *service model* for "channel" transporting datagrams from sender to receiver?**

example services for individual datagrams:

- **Guaranteed delivery**
- **Guaranteed delivery with less than 40 msec delay**

example services for a flow of datagrams:

- **In-order datagram delivery**
- **Guaranteed minimum bandwidth to flow**
- **Restrictions on changes in inter-packet spacing**

Network layer service models:

Network Architecture	Service Model	Guarantees ?				Congestion feedback
		Bandwidth	Loss	Order	Timing	
Internet	best effort	none	no	no	no	no (inferred via loss)
ATM	CBR	constant rate	yes	yes	yes	no congestion
ATM	VBR	guaranteed rate	yes	yes	yes	no congestion
ATM	ABR	guaranteed minimum	no	yes	no	yes
ATM	UBR	none	no	yes	no	no

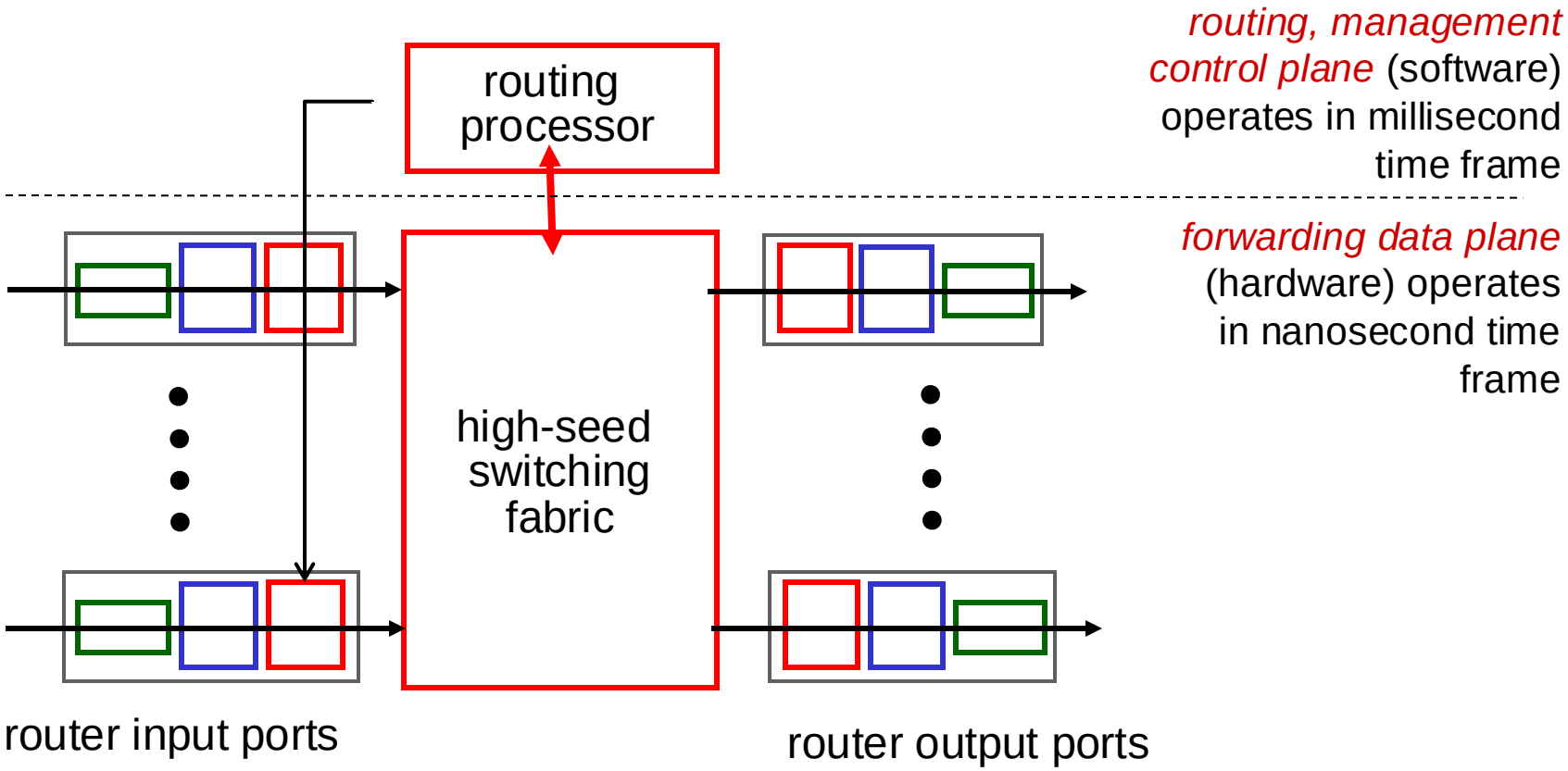
* Asynchronous Transfer Mode（ATM）

Lecture 6 – Network Layer (1)

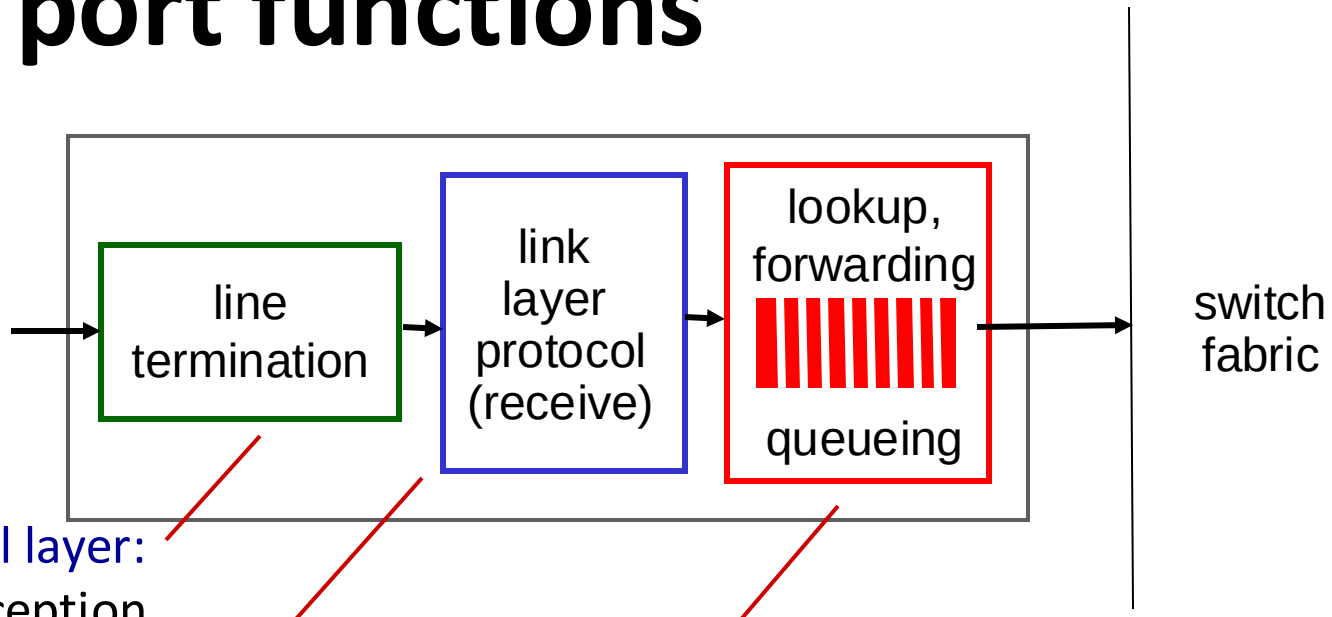
- **Roadmap**
  1. Overview of Network layer
  2. Router
  3. Internet Protocol

Router architecture overview

- **High-level view of generic router architecture:**



Input port functions



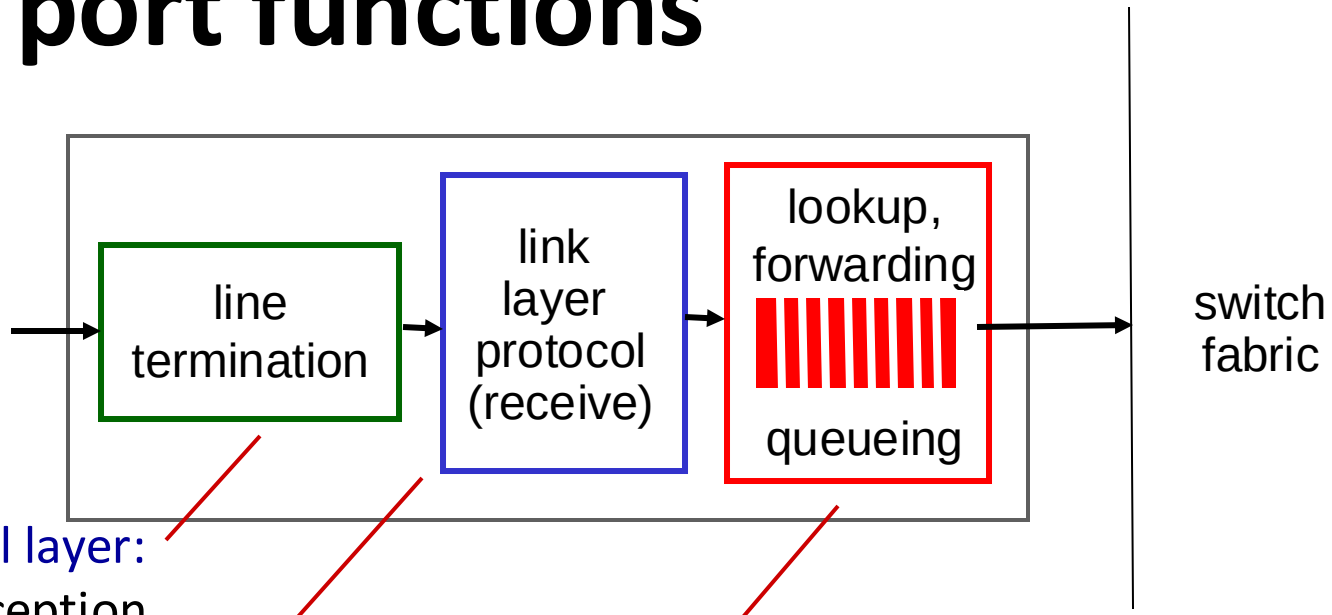
physical layer:
bit-level reception

data link layer:
e.g., Ethernet
see chapter 6

Decentralized switching:

- using header field values, lookup output port using forwarding table in input port memory *("match plus action")*
- goal: complete input port processing at 'line speed'
- queuing: if datagrams arrive faster than forwarding rate into switch fabric

Input port functions



physical layer:
bit-level reception

data link layer:
e.g., Ethernet
see chapter 6

Decentralized switching:

- using header field values, lookup output port using forwarding table in input port memory *("match plus action")*
- *destination-based forwarding:* forward based only on destination IP address (traditional)
- *generalized forwarding:* forward based on any set of header field values

Destination-based forwarding

forwarding table

	Destination Address Range	Link Interface
200.23.16.0 200.23.23.255	**11001000 00010111 00010000 00000000** through **11001000 00010111 00010111 11111111**	0
200.23.24.0 200.23.24.255	**11001000 00010111 00011000 00000000** through **11001000 00010111 00011000 11111111**	1
200.23.25.0 200.23.31.255	**11001000 00010111 00011001 00000000** through **11001000 00010111 00011111 11111111**	2
	otherwise	3

Q: but what happens if ranges don't divide up so nicely?

Longest prefix matching

- **When looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.**

Destination Address Range	Link interface
11001000 00010111 00010*** *********	0
11001000 00010111 00011000 *********	1
11001000 00010111 00011*** *********	2
otherwise	3

Examples:

DA: 11001000 00010111 00010110 10100001 which interface?

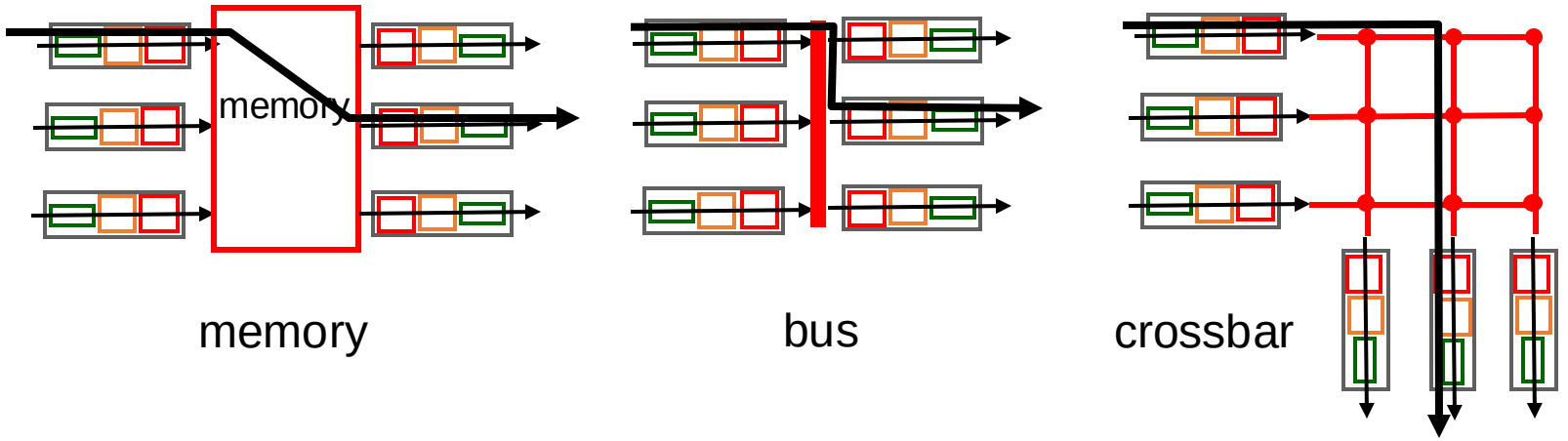
DA: 11001000 00010111 00011000 10101010 which interface?

Longest prefix matching

- **We'll see *why* longest prefix matching is used shortly, when we study addressing**
- **Longest prefix matching: often performed using ternary content addressable memories (TCAMs)**
  - *Content addressable:* present address to TCAM: retrieve address in one clock cycle, regardless of table size
  - Cisco Catalyst: can up ~1M routing table entries in TCAM

Switching fabrics

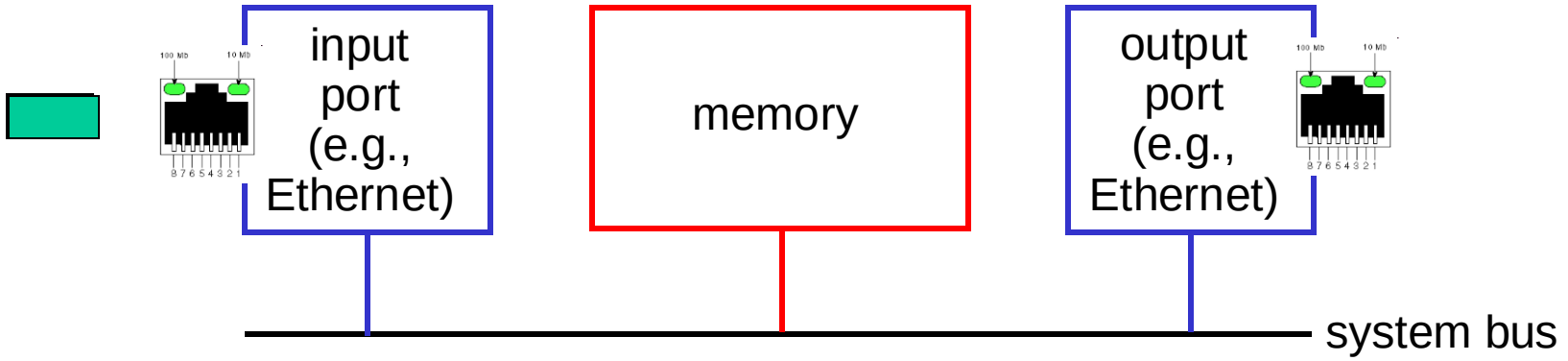
- **Transfer packet from input buffer to appropriate output buffer**
- **Switching rate: rate at which packets can be transfer from inputs to outputs**
  - Often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate desirable
- **Three types of switching fabrics:**



Switching via memory

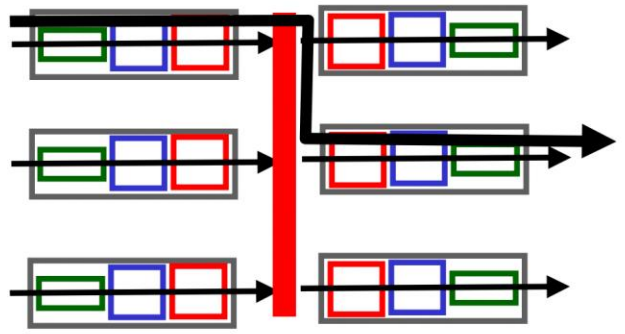
First generation routers:

- Traditional computers with switching under direct control of CPU
- Packet copied to system's memory
- Speed limited by memory bandwidth (2 bus crossings per datagram)



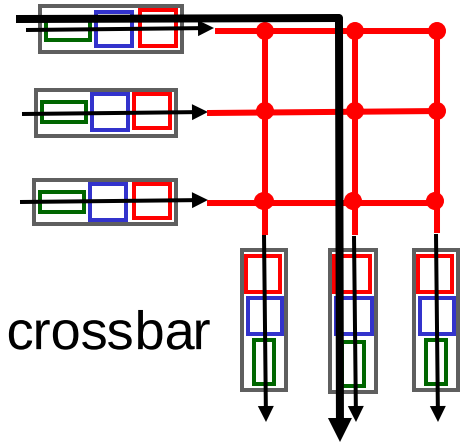
Switching via a bus

- Datagrams from input port memory to output memory via a shared bus
- Bus contention: switching speed limited by bus bandwidth
- Cisco 5600: 32 Gbps bus



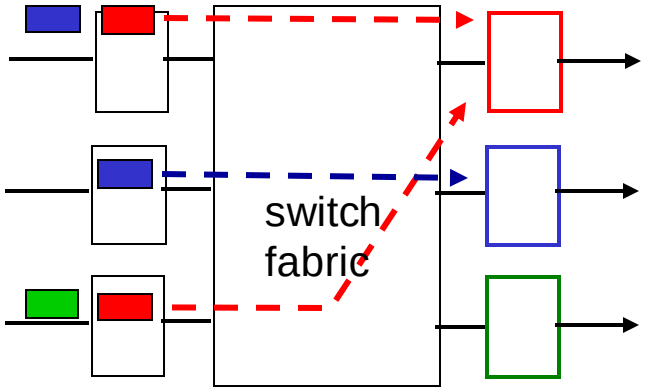
Switching via interconnection network

- Overcome bus bandwidth limitations
- Banyan networks, crossbar, other interconnection nets initially developed to connect processors in multiprocessor
- Advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
- Cisco 12000: switches 60 Gbps through the interconnection network

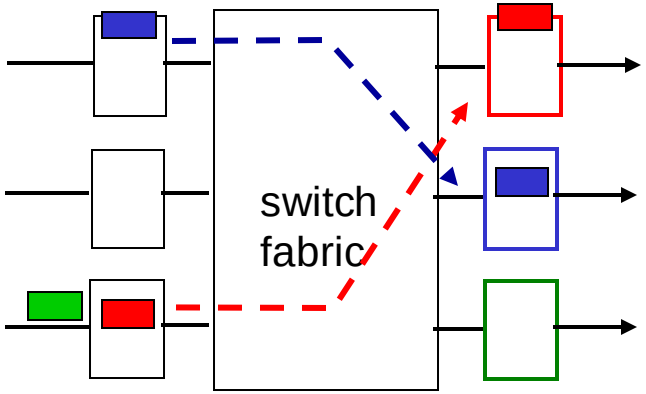


Input port queuing

- **Fabric slower than input ports combined -> queueing may occur at input queues**
  - *queueing delay and loss due to input buffer overflow!*
- **Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward**

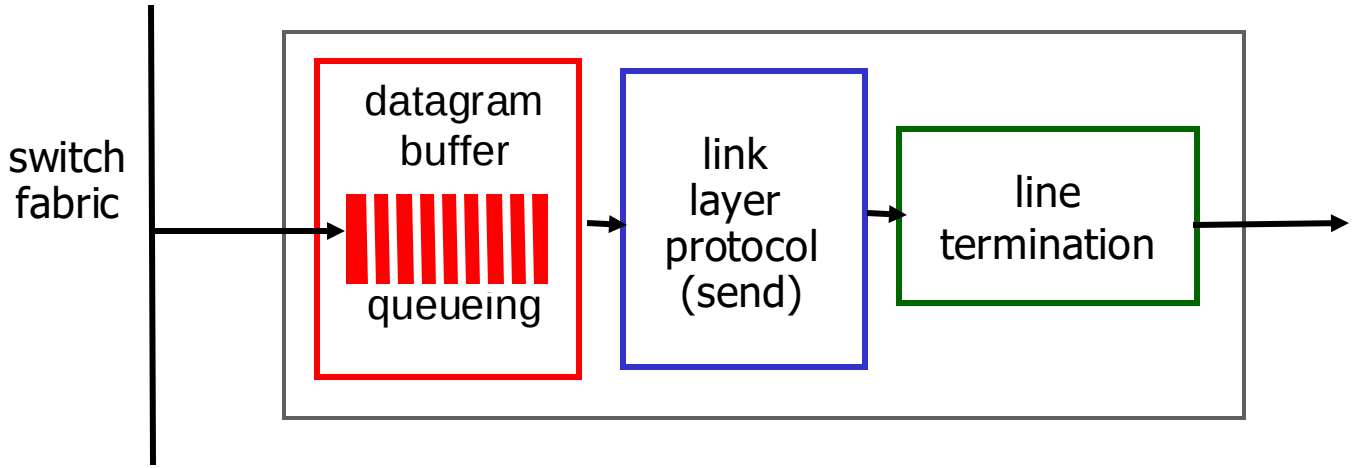


output port contention:
only one red datagram can be transferred.
lower red packet is blocked



one packet time later:
green packet
experiences HOL
blocking

Output ports

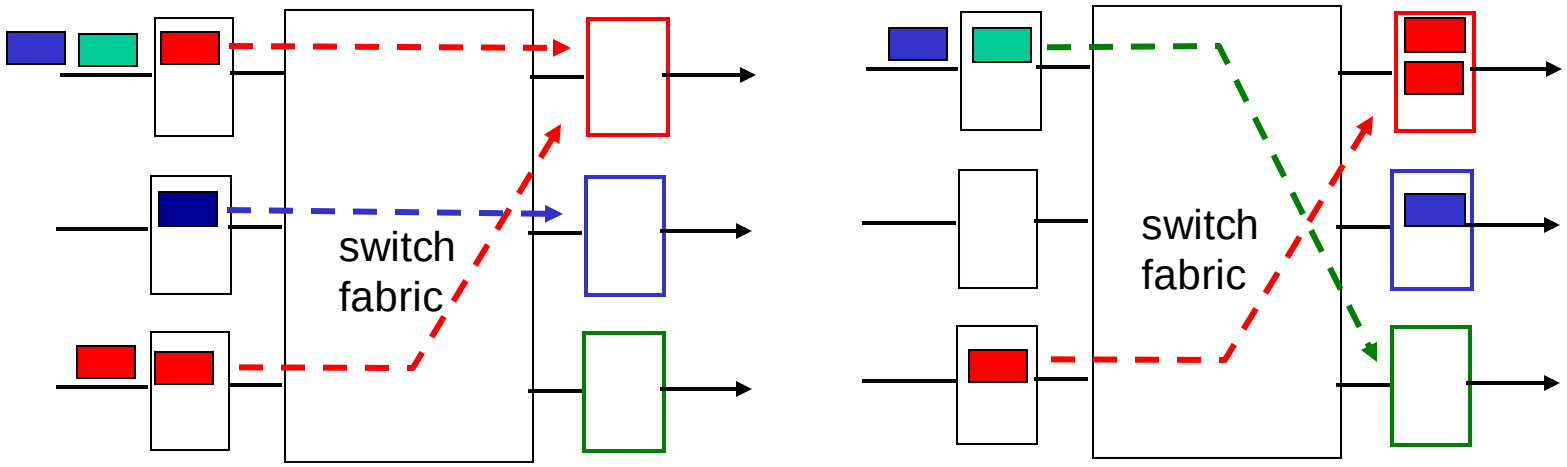


- *Buffering* required when datagram from fabric faster than the transm rate
- *Scheduling discipline* chooses datagrams for transmission

Datagram (packets) can be lost due to congestion, lack of buffers

Priority scheduling – who gets best performance, network neutrality

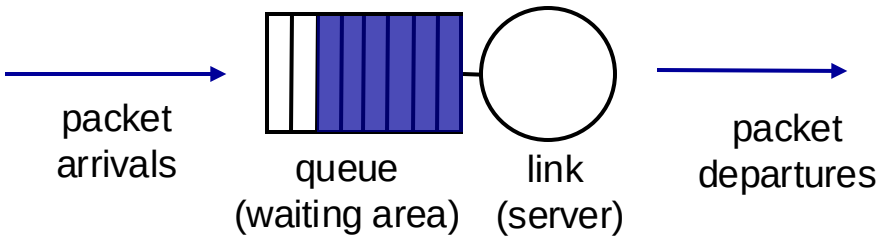
Output port queueing



- Buffering when arrival rate via switch exceeds output line speed
- *Queueing (delay) and loss due to output port buffer overflow!*

Scheduling mechanisms

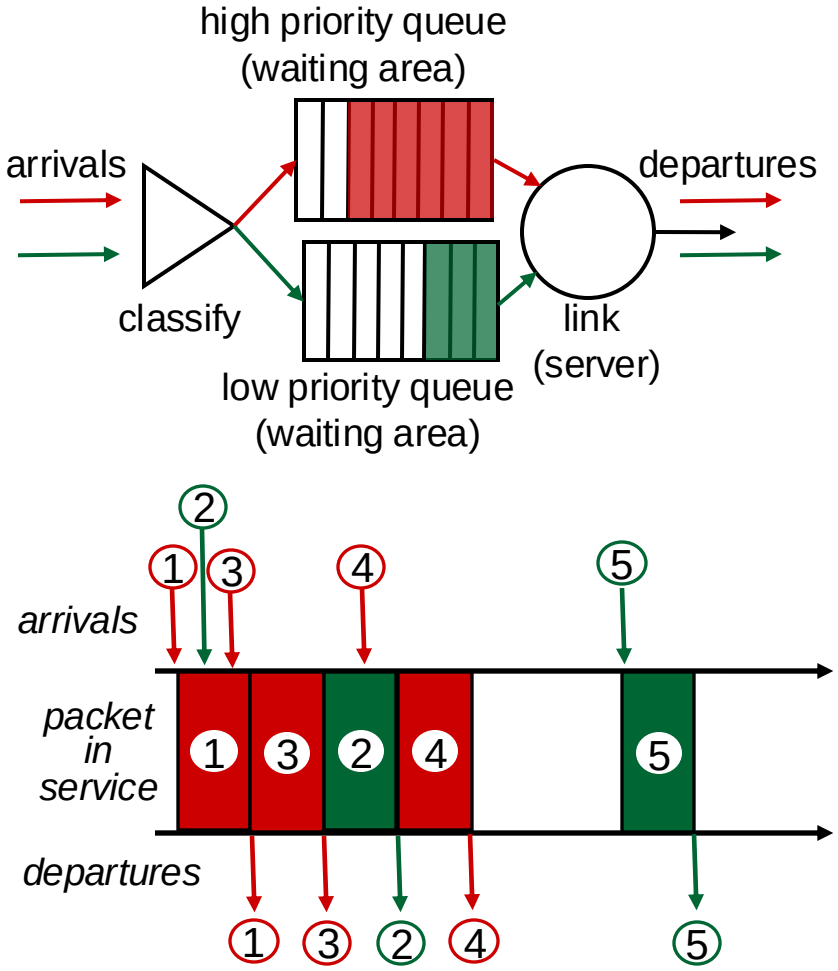
- ***Scheduling:*** **choose next packet to send on link**
- ***FIFO (first in first out) scheduling:*** **send in order of arrival to queue**
  - Real-world example?
  - *Discard policy:* if packet arrives to full queue: who to discard?
    - *Tail drop:* drop arriving packet
    - *Priority:* drop/remove on priority basis
    - *Random:* drop/remove randomly



Scheduling policies: priority

Priority scheduling: **send highest priority queued packet**

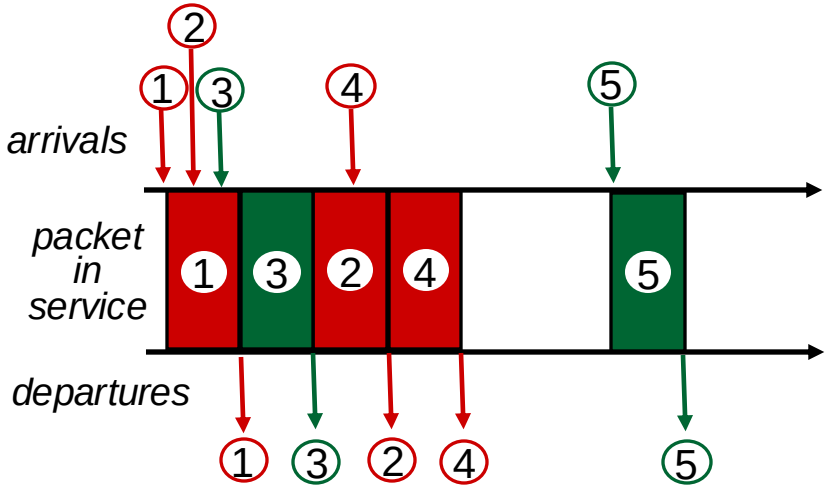
- **Multiple *classes*, with different priorities**
  - Class may depend on marking or other header info, e.g. IP source/dest, port numbers, etc.
  - real world example?



Scheduling policies: still more

Round Robin (RR) scheduling:

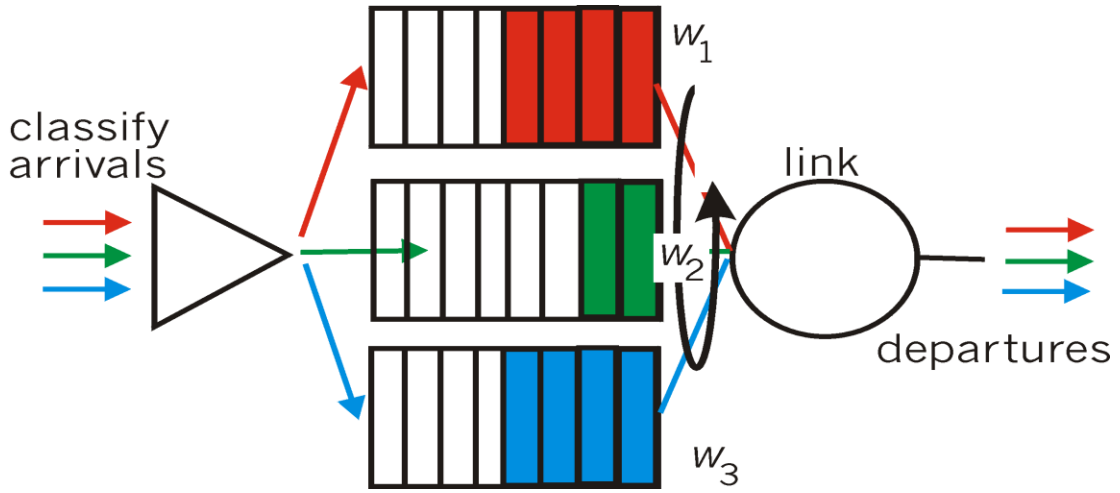
- Multiple classes
- Cyclically scan class queues, sending one complete packet from each class (if available)
- Real world example?



Scheduling policies: still more

Weighted Fair Queuing (WFQ):

- **Generalized Round Robin**
- **Each class gets weighted amount of service in each cycle**
- **Real-world example?**

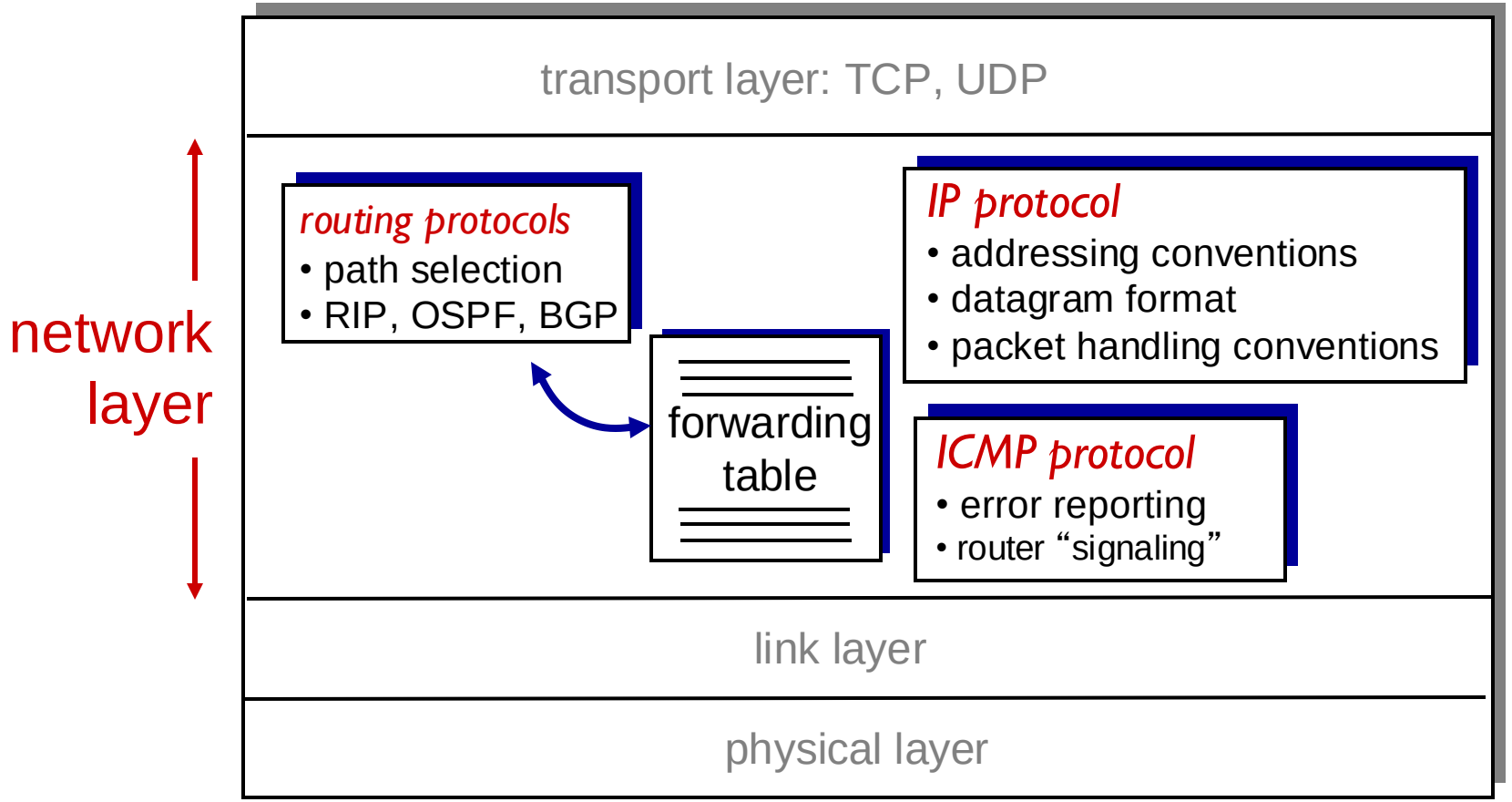


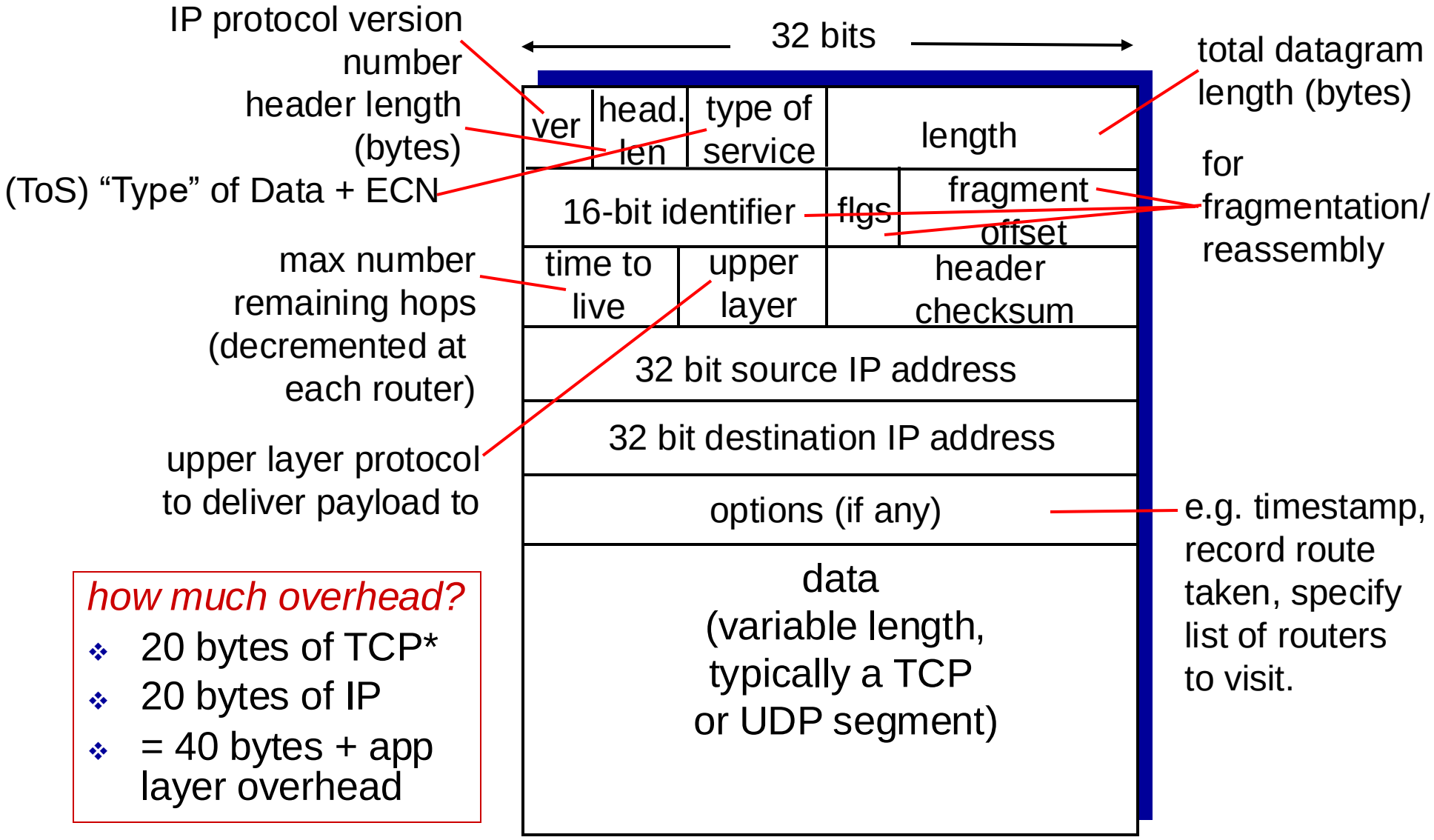
Lecture 6 – Network Layer (1)

- **Roadmap**
  1. Overview of Network layer
  2. Router
  3. Internet Protocol

The Internet network layer

Host, router network layer functions:





how much overhead?

- 20 bytes of TCP*
- 20 bytes of IP
- = 40 bytes + app layer overhead

IP datagram format

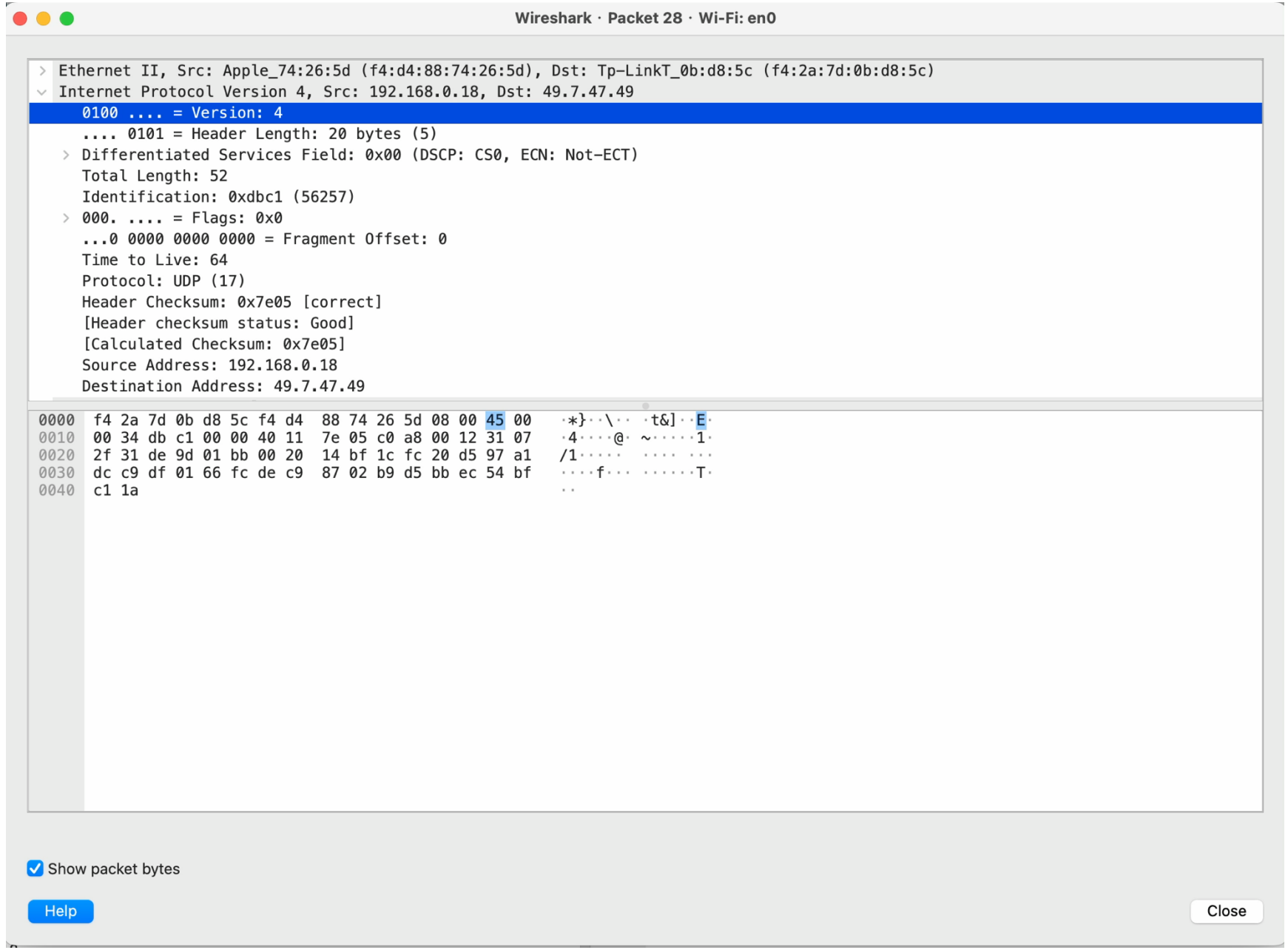
*sometimes the optional header will be used, the overhead of TCP will be more.

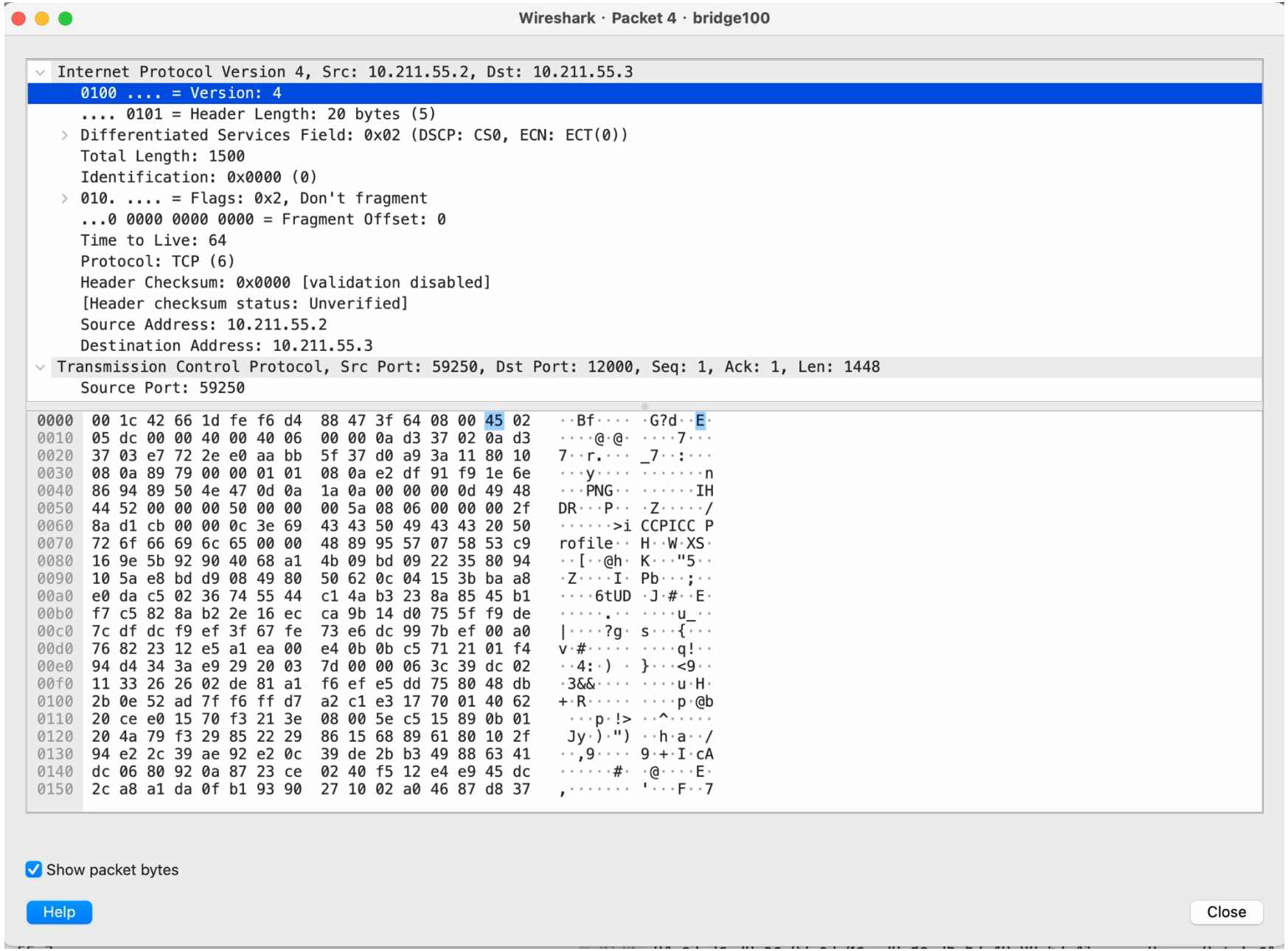
ToS: Type of service

7	6	5	4	3	2	1	0
DSCP						ECN	

Differentiated Services Code Point (DSCP)

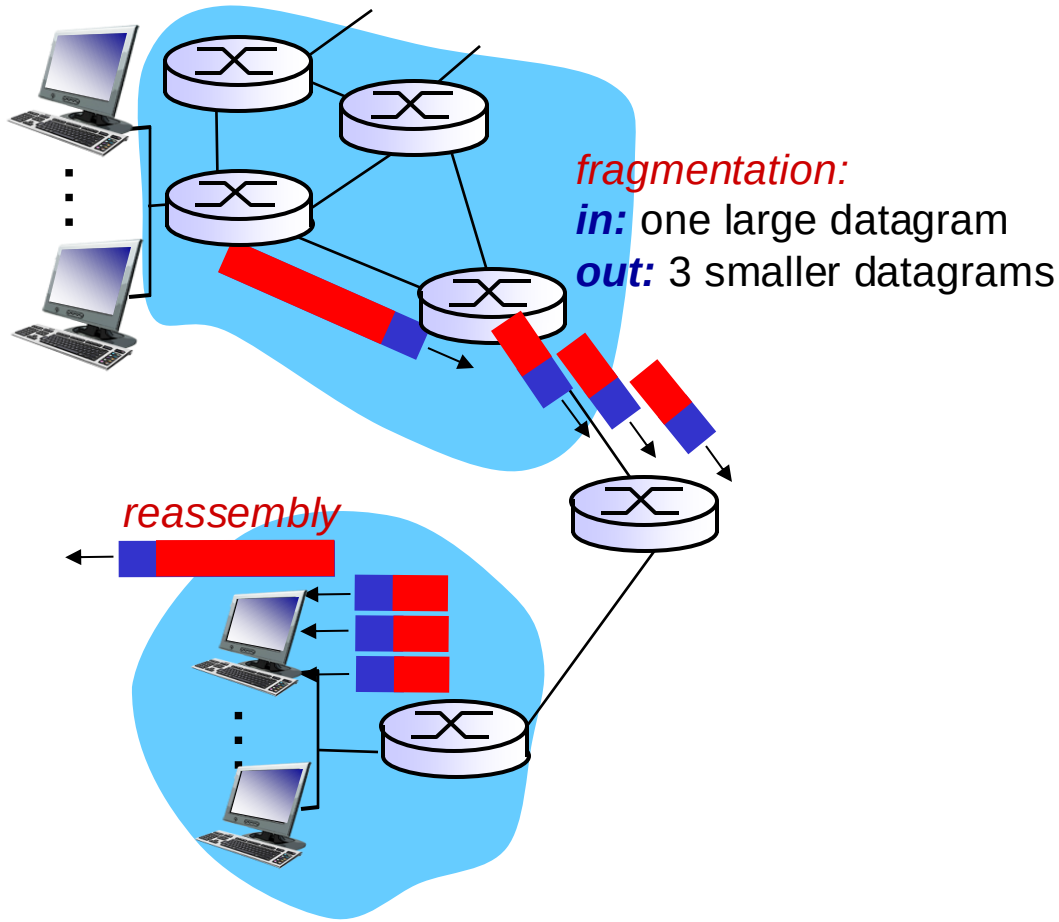
Service class	DSCP Name	DSCP Value	Examples of application
Standard	CS0 (DF)	0	
Low–priority data	CS1	8	File transfer (FTP, SMB)
Network operations, administration and management (OAM)	CS2	16	SNMP, SSH, Ping, Telnet, syslog
Broadcast video	CS3	24	RTSP broadcast TV streaming of live audio and video events video surveillance video–on–demand
Real–time interactive	CS4	32	Gaming, low priority video conferencing
Signaling	CS5	40	Peer–to–peer (SIP, H.323, H.248), NTP
Network control	CS6	48	Routing protocols (OSPF, BGP, ISIS, RIP)
Reserved for future use	CS7	56	


Wireshark · Packet 28 · Wi-Fi: en0
Ethernet II, Src: Apple_74:26:5d (f4:d4:88:74:26:5d), Dst: Tp-LinkT_0b:d8:5c (f4:2a:7d:0b:d8:5c)
Internet Protocol Version 4, Src: 192.168.0.18, Dst: 49.7.47.49
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 52
Identification: 0xdbc1 (56257)
000. .... = Flags: 0x0
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 64
Protocol: UDP (17)
Header Checksum: 0x7e05 [correct]
[Header checksum status: Good]
[Calculated Checksum: 0x7e05]
Source Address: 192.168.0.18
Destination Address: 49.7.47.49
0000 f4 2a 7d 0b d8 5c f4 d4 88 74 26 5d 08 00 45 00
0010 00 34 db c1 00 00 40 11 7e 05 c0 a8 00 12 31 07
0020 2f 31 de 9d 01 bb 00 20 14 bf 1c fc 20 d5 97 a1
0030 dc c9 df 01 66 fc de c9 87 02 b9 d5 bb ec 54 bf
0040 c1 1a
Show packet bytes
Help
Close


Wireshark · Packet 4 · bridge100
Internet Protocol Version 4, Src: 10.211.55.2, Dst: 10.211.55.3
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x02 (DSCP: CS0, ECN: ECT(0))
Total Length: 1500
Identification: 0x0000 (0)
010. .... = Flags: 0x2, Don't fragment
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 64
Protocol: TCP (6)
Header Checksum: 0x0000 [validation disabled]
[Header checksum status: Unverified]
Source Address: 10.211.55.2
Destination Address: 10.211.55.3
Transmission Control Protocol, Src Port: 59250, Dst Port: 12000, Seq: 1, Ack: 1, Len: 1448
Source Port: 59250
0000 00 1c 42 66 1d fe f6 d4 88 47 3f 64 08 00 45 02
0010 05 dc 00 00 40 00 40 06 00 00 0a d3 37 02 0a d3
0020 37 03 e7 72 2e e0 aa bb 5f 37 d0 a9 3a 11 80 10
0030 08 0a 89 79 00 00 01 01 08 0a e2 df 91 f9 1e 6e
0040 86 94 89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48
0050 44 52 00 00 00 50 00 00 00 5a 08 06 00 00 00 2f
0060 8a d1 cb 00 00 0c 3e 69 43 43 50 49 43 43 20 50
0070 72 6f 66 69 6c 65 00 00 48 89 95 57 07 58 53 c9
0080 16 9e 5b 92 90 40 68 a1 4b 09 bd 09 22 35 80 94
0090 10 5a e8 bd d9 08 49 80 50 62 0c 04 15 3b ba a8
00a0 e0 da c5 02 36 74 55 44 c1 4a b3 23 8a 85 45 b1
00b0 f7 c5 82 8a b2 2e 16 ec ca 9b 14 d0 75 5f f9 de
00c0 7c df dc f9 ef 3f 67 fe 73 e6 dc 99 7b ef 00 a0
00d0 76 82 23 12 e5 a1 ea 00 e4 0b 0b c5 71 21 01 f4
00e0 94 d4 34 3a e9 29 20 03 7d 00 00 06 3c 39 dc 02
00f0 11 33 26 26 02 de 81 a1 f6 ef e5 dd 75 80 48 db
0100 2b 0e 52 ad 7f f6 ff d7 a2 c1 e3 17 70 01 40 62
0110 20 ce e0 15 70 f3 21 3e 08 00 5e c5 15 89 0b 01
0120 20 4a 79 f3 29 85 22 29 86 15 68 89 61 80 10 2f
0130 94 e2 2c 39 ae 92 e2 0c 39 de 2b b3 49 88 63 41
0140 dc 06 80 92 0a 87 23 ce 02 40 f5 12 e4 e9 45 dc
0150 2c a8 a1 da 0f b1 93 90 27 10 02 a0 46 87 d8 37
Show packet bytes
Help
Close

IP fragmentation, reassembly

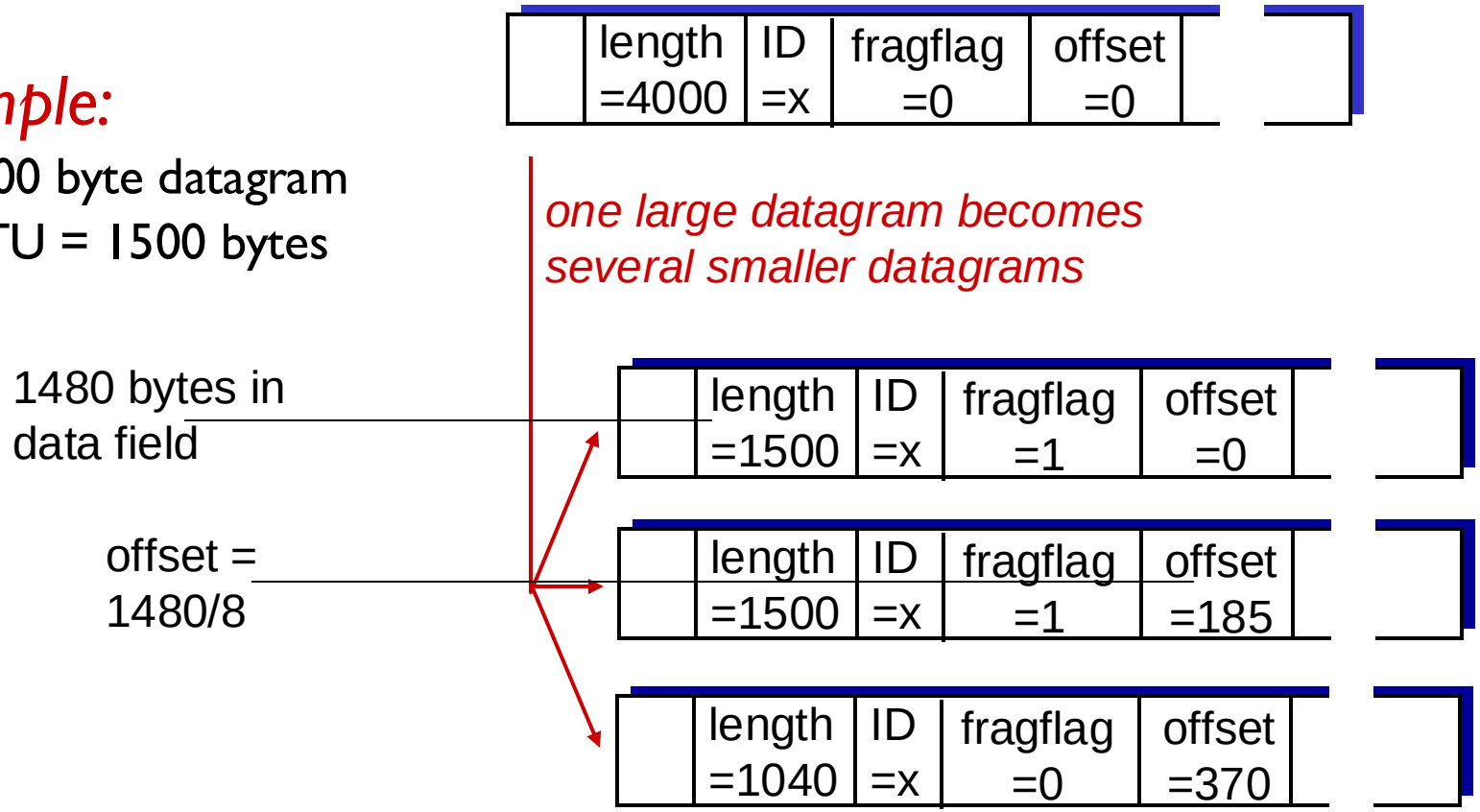
- **Network links have MTU (max.transfer size) - largest possible link-level frame**
  - Different link types, different MTUs
- **Large IP datagram divided ("fragmented") within net**
  - One datagram becomes several datagrams
  - "Reassembled" only at final destination
  - IP header bits used to identify, order related fragments



IP fragmentation, reassembly

example:

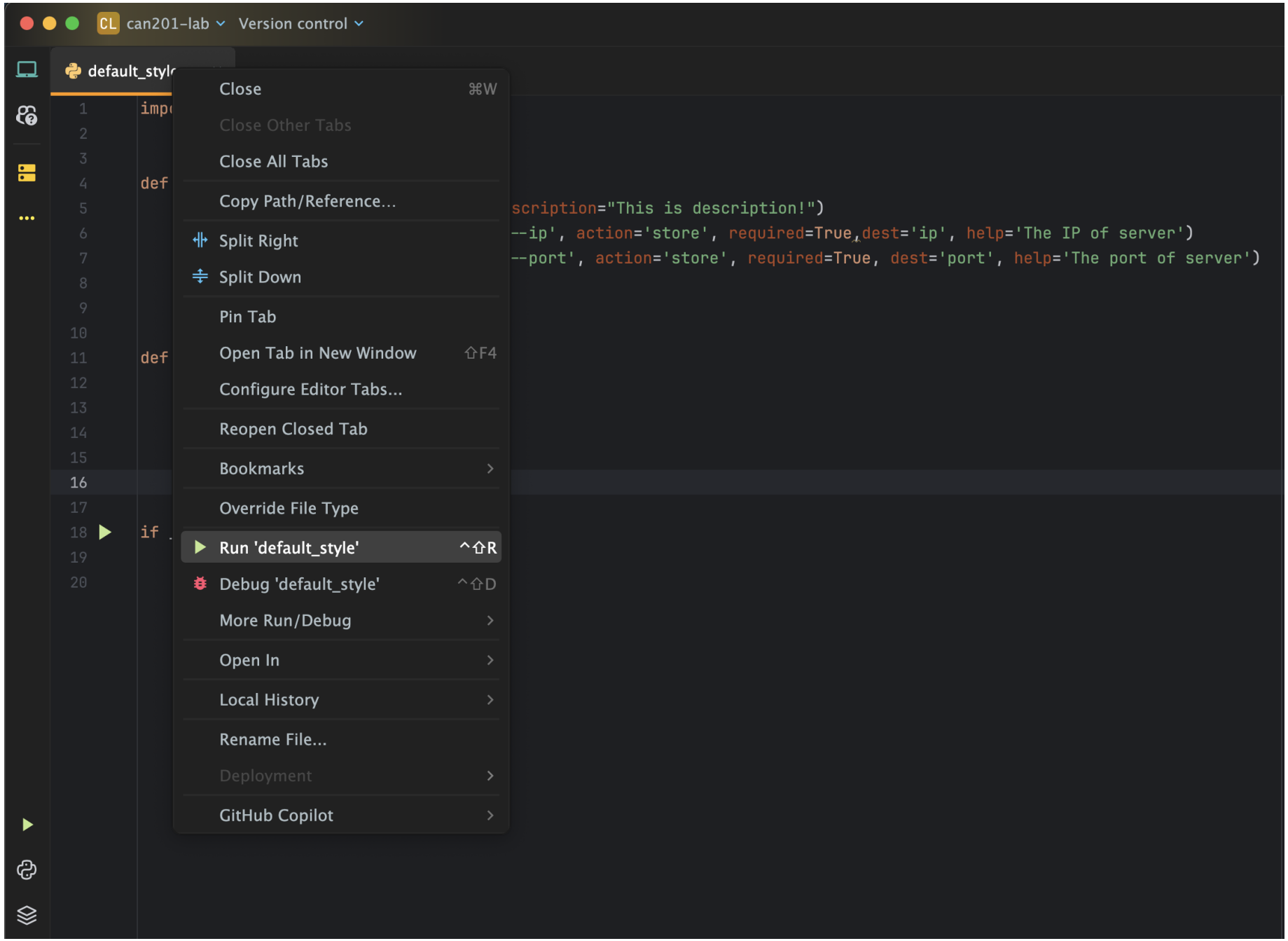
- 4000 byte datagram
- MTU = 1500 bytes

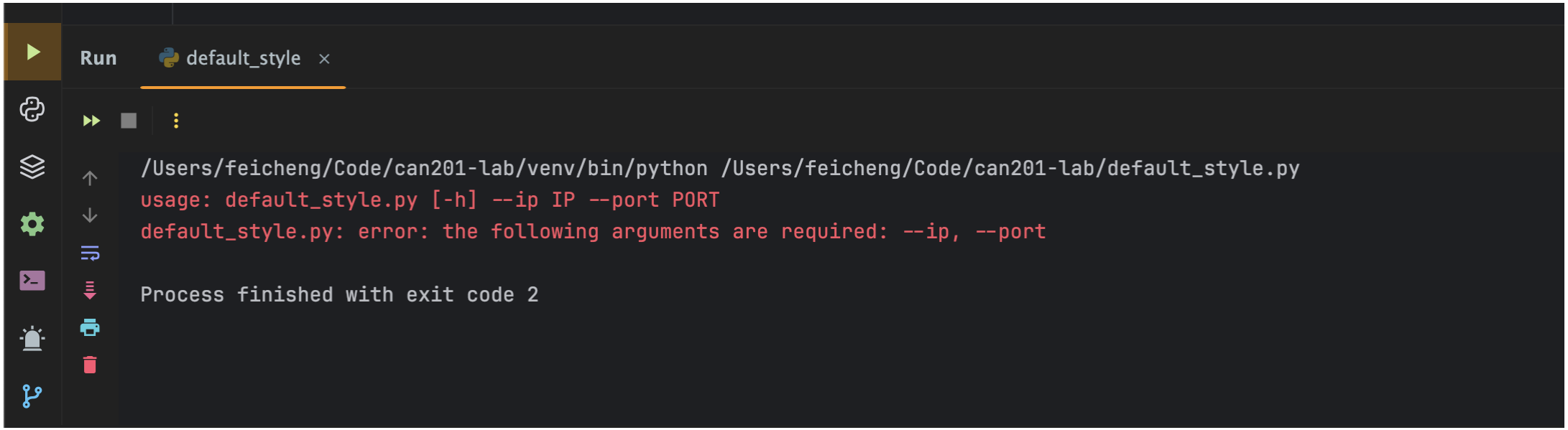


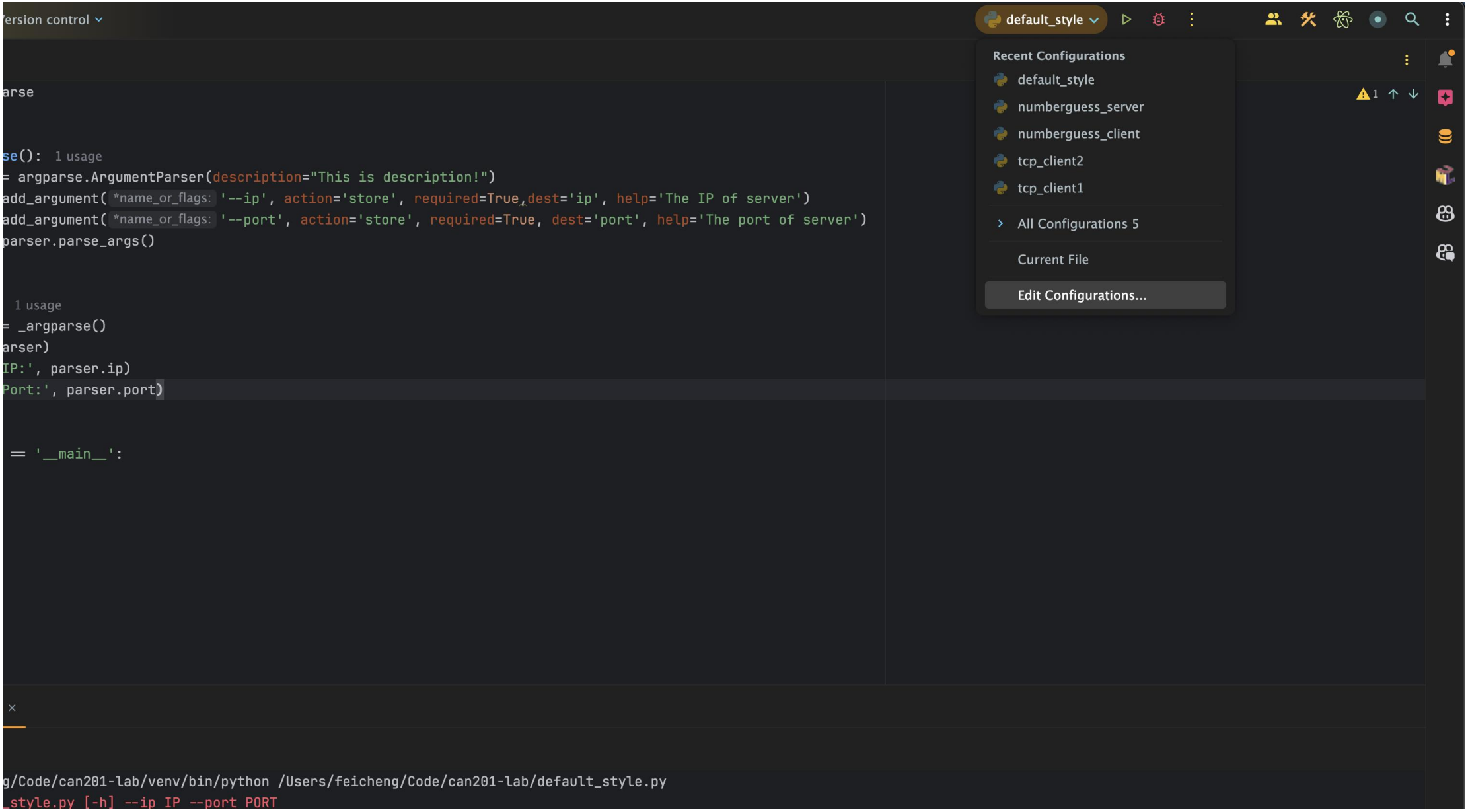
In Class Test 1 - Preview

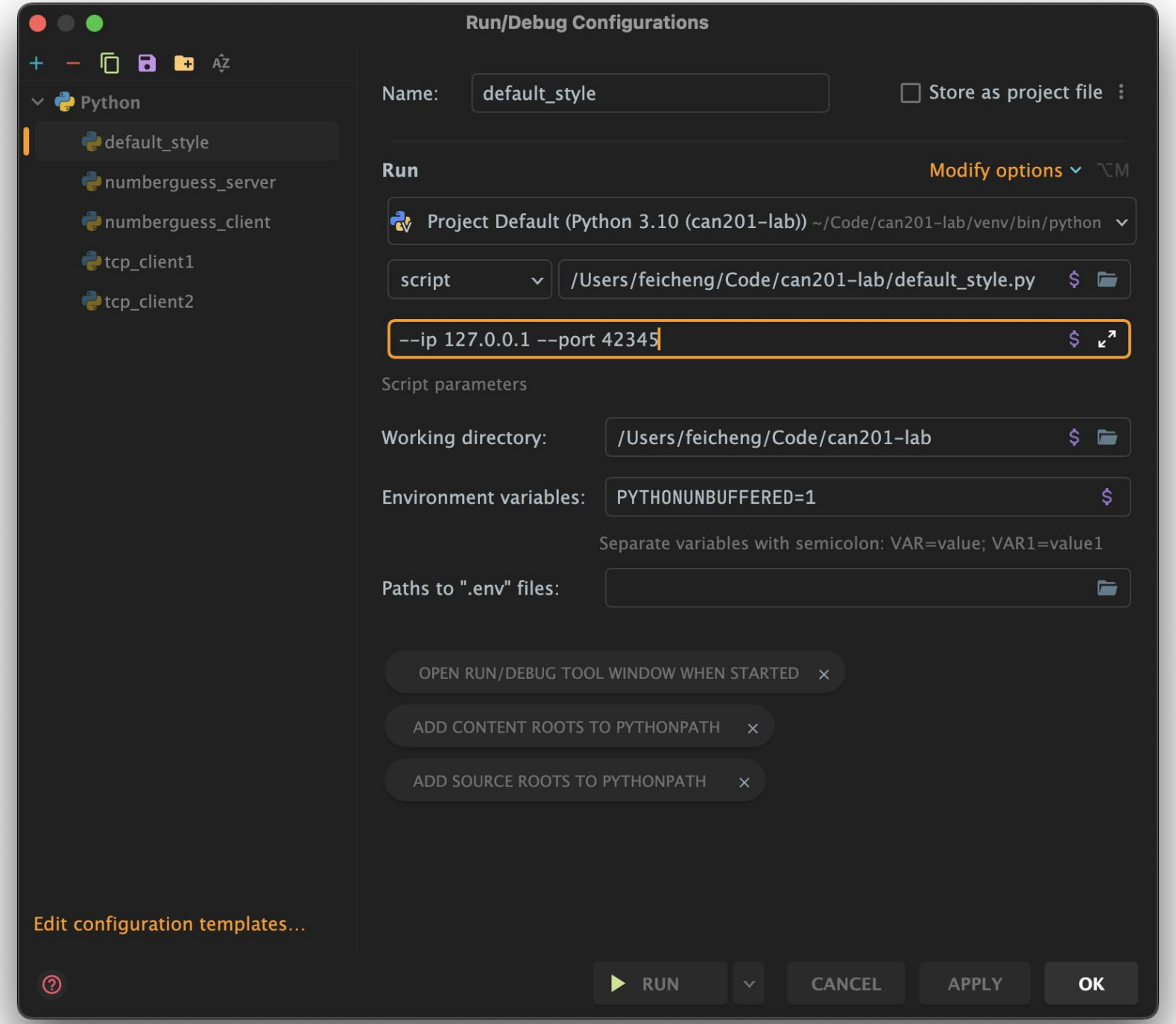
```python
import argparse


def _argparse():
    parser = argparse.ArgumentParser(description="This is description!")
    parser.add_argument('--ip', action='store', required=True, dest='ip', help='The IP of server')
    parser.add_argument('--port', action='store', required=True, dest='port', help='The port of server')
    return parser.parse_args()


def main():
    parser = _argparse()
    print(parser)
    print('IP:', parser.ip)
    print('Port:', parser.port)


if __name__ == '__main__':
    main()
```

https://box.xjtlu.edu.cn/f/947b9025b5e84f98b73b/


CL can201-lab
Version control
default_style
Close ⌘W
Close Other Tabs
Close All Tabs
Copy Path/Reference...
Split Right
Split Down
Pin Tab
Open Tab in New Window ⇧F4
Configure Editor Tabs...
Reopen Closed Tab
Bookmarks
Override File Type
Run 'default_style' ^⇧R
Debug 'default_style' ^⇧D
More Run/Debug
Open In
Local History
Rename File...
Deployment
GitHub Copilot
scription="This is description!")
--ip', action='store', required=True,dest='ip', help='The IP of server')
--port', action='store', required=True, dest='port', help='The port of server')


Run
default_style
/Users/feicheng/Code/can201-lab/venv/bin/python /Users/feicheng/Code/can201-lab/default_style.py
usage: default_style.py [-h] --ip IP --port PORT
default_style.py: error: the following arguments are required: --ip, --port
Process finished with exit code 2


default_style
Recent Configurations
default_style
numberguess_server
numberguess_client
tcp_client2
tcp_client1
All Configurations 5
Current File
Edit Configurations...
arse
se(): 1 usage
= argparse.ArgumentParser(description="This is description!")
add_argument(*name_or_flags: '--ip', action='store', required=True,dest='ip', help='The IP of server')
add_argument(*name_or_flags: '--port', action='store', required=True, dest='port', help='The port of server')
parser.parse_args()
1 usage
= _argparse()
arser)
IP:', parser.ip)
Port:', parser.port)
== '__main__':
g/Code/can201-lab/venv/bin/python /Users/feicheng/Code/can201-lab/default_style.py
_style.py [-h] --ip IP --port PORT


Run/Debug Configurations
Python
default_style
numberguess_server
numberguess_client
tcp_client1
tcp_client2
Edit configuration templates...
Name:
default_style
Store as project file
Run
Modify options
Project Default (Python 3.10 (can201-lab)) ~/Code/can201-lab/venv/bin/python
script
/Users/feicheng/Code/can201-lab/default_style.py
--ip 127.0.0.1 --port 42345
Script parameters
Working directory:
/Users/feicheng/Code/can201-lab
Environment variables:
PYTHONUNBUFFERED=1
Separate variables with semicolon: VAR=value; VAR1=value1
Paths to ".env" files:
OPEN RUN/DEBUG TOOL WINDOW WHEN STARTED
ADD CONTENT ROOTS TO PYTHONPATH
ADD SOURCE ROOTS TO PYTHONPATH
RUN
CANCEL
APPLY
OK


Run
default_style
/Users/feicheng/Code/can201-lab/venv/bin/python /Users/feicheng/Code/can201-lab/default_style.py --ip 127.0.0.1 --port 42345
Namespace(ip='127.0.0.1', port='42345')
IP: 127.0.0.1
Port: 42345
Process finished with exit code 0

“Multi-client” UDP server

```
# server side
from socket import *

server_port = 12000
server_socket = socket(AF_INET, SOCK_DGRAM)
server_socket.bind(('', server_port))

print('The server is ready!')

records = []

while True:
    message, client_address = server_socket.recvfrom(20480)
    print(client_address, 'said', message.decode())
    records.append([client_address, message.decode()])
    print(records)
    modified_message = message.decode().upper()
    server_socket.sendto(modified_message.encode(), client_address)
```

UDP client

```
# client side
from socket import *

server_hostname = '127.0.0.1'
server_port = 12000

client_socket = socket(AF_INET, SOCK_DGRAM)

message = input('Input a sentence:')

client_socket.sendto(message.encode(), (server_hostname, server_port))
modified_message, server_address = client_socket.recvfrom(20480)
print(modified_message.decode(), server_address)
client_socket.close()
```

https://box.xjtlu.edu.cn/f/d6e71b8860254152995d/

“Multi-client” TCP server

```
from socket import *
import threading

server_port = 12002
server_socket = socket(AF_INET, SOCK_STREAM)

server_socket.bind(('', server_port))
server_socket.listen(10)

print('TCP server is listening!')

records = []  # A global list to store all the records!!!

def TCP_processor(connection_socket, address):
    global records
    print(address, ' connected')
    while True:
        try:
            sentence = connection_socket.recv(20480).decode()
            if sentence == '':
                break
            print(address, ' said ', sentence)
            records.append([address, sentence])
            print(records)
            modified_message = sentence.upper()
            connection_socket.send(modified_message.encode())
        except Exception as ex:
            break
    print(address, ' disconnected')
    connection_socket.close()


while True:
    try:
        connection_socket, address = server_socket.accept()
        th = threading.Thread(target=TCP_processor, args=(connection_socket, address))
        th.start()
    except Exception as ex:
        print(ex)
```

TCP Client

```
from socket import *

server_hostname = '127.0.0.1'
server_port = 12002
client_socket = socket(AF_INET, SOCK_STREAM)
client_socket.connect((server_hostname, server_port))

while True:
    sentence = input('Input a sentence')
    if sentence == '':
        break
    client_socket.send(sentence.encode())
    modified_message = client_socket.recv(20480)
    print(modified_message.decode())
client_socket.close()
```

https://box.xjtlu.edu.cn/f/f4a8bf41ab2b4fa8b97e/

More info…

- **The specifications for 6 sessions are slightly different.**
- **The submission link and specification will be released 10 min earlier.**
- **Please pay more attention to the difference.**
- **Total points: 5**
- **If you cannot submit during your lab session (+15 mins): -1**
  - eg. for the lab session start from 9:00, this deadline will be 11:15.
- **If you cannot submit within 24 hours: -1 more (total -2)**
  - eg. for the lab session start from 9:00, this deadline will be 11:00 tomorrow.

Thanks.