CSE 461

Final Review

Transport Layer

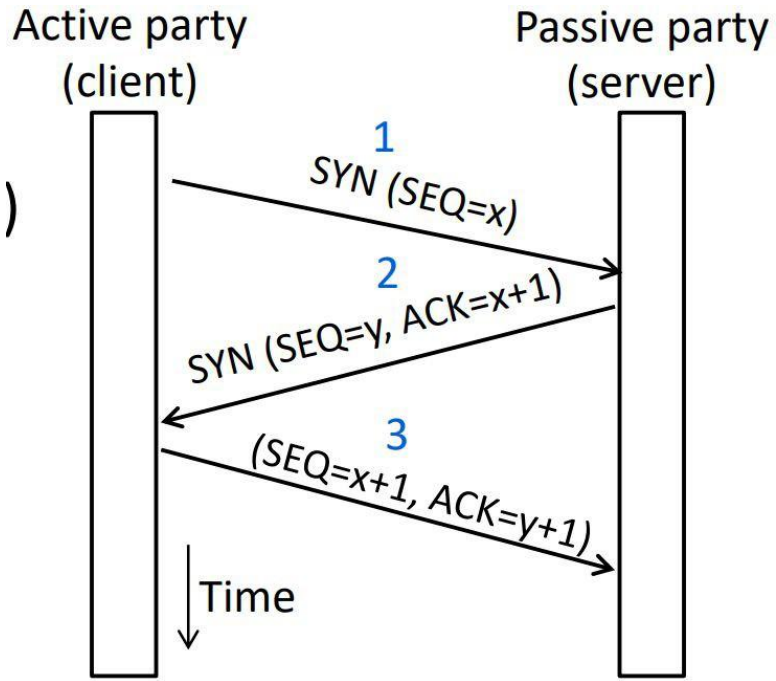
- Provides end-to-end data delivery / connectivity to applications

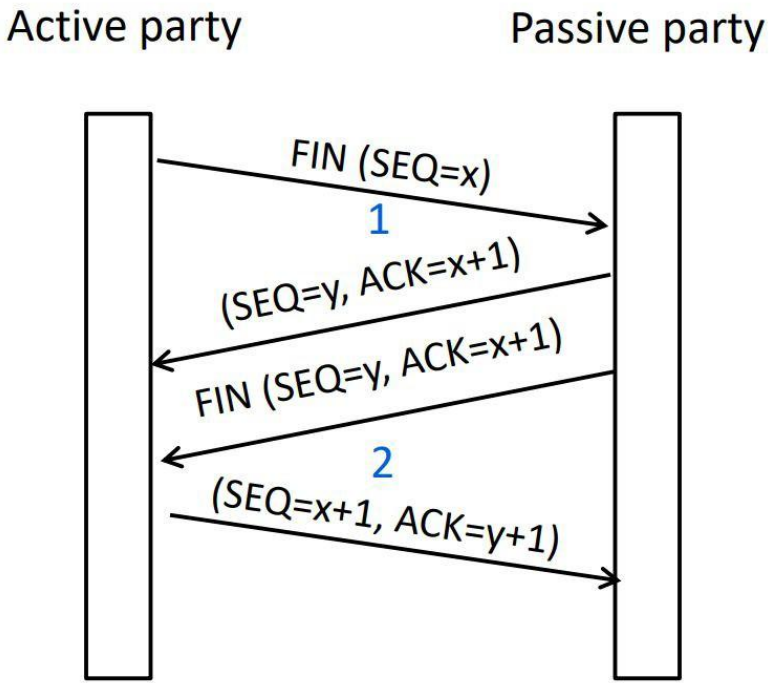
- Transport Layer Services
  - Datagrams (UDP): Unreliable Messages
  - Streams (TCP): Reliable Bytestreams

TCP vs UDP

TCP (Streams)	UDP (Datagrams)
Connections	Datagrams
Bytes are delivered once, reliably, and in order	Messages may be lost, reordered, duplicated
Arbitrary length content	Limited message size
Flow control matches sender to receiver	Can send regardless of receiver state
Congestion control matches sender to network	Can send regardless of network state

TCP Connection Establishment and Release





Flow Control - Sliding Window Protocol

- Instead of stop-and-wait, sends W packets per 1 RTT
  - To fill network path, W = B * RTT / packet_size
- Receiver sends ACK upon receiving packets
  - Go-Back-N (similar to project 1 stage b): not efficient
  - **Selective Repeat**
    - Receiver passes data to app in order, and buffers out-of-order segments to reduce retransmissions
    - ACK conveys highest in-order segment
      - As well as hints about out-of-order segments
- **Selective Retransmission** on sender's side

Exercise: Stop-and-wait vs Selective Repeat

- Suppose a sender A transmits 5 packets to receiver B, where RTT = 20 ms and packet timeout = 30 ms, and assume the 3rd packet is dropped. Compare the time taken for A to send all 5 packets and receive their Acks using Stop-and-Wait vs Selective Repeat (W = 3).

Exercise: Stop-and-wait vs Selective Repeat

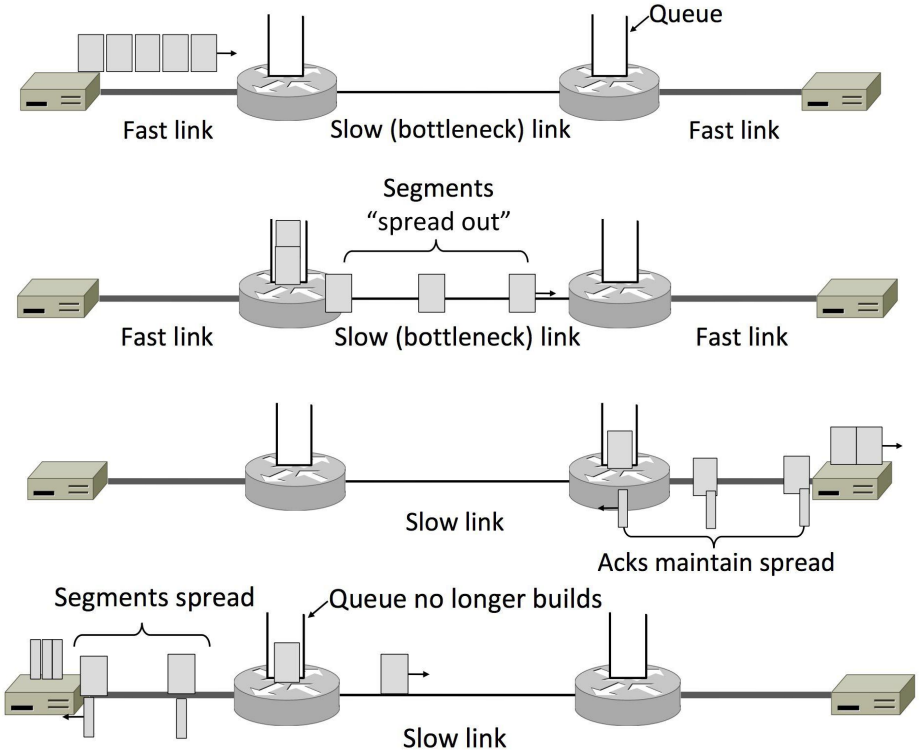
Suppose a sender A transmits 5 packets to receiver B, where RTT = 20 ms and packet timeout = 30 ms, and assume the 3rd packet is dropped. Compare the time taken for A to send all 5 packets and receive their Acks using Stop-and-Wait vs Selective Repeat (W = 3).

Solution:

ARQ: sends one packet at a time. So first 2 packets take 2 * 20 = 40 ms to be ACK'd. When the 3rd packet is sent and dropped, the sender waits until its timeout timer fires (30 ms) before retransmitting packet #3. From there, retransmits 3 and sends 4 and 5. Total time = 2 * 20 + 30 + 3 * 20 = 130 ms.

Selective Repeat: W = 3, so sends first 3 packets together and receives acks for #1 and #2, but not for #3. Window is advanced by 2, so sends 4 and 5 in second RTT. At this point when 4 and 5 are sent, timer of #3 has 10 ms before it fires. During second RTT, timer for packet #3 fires, so retransmits 3. (When 4, 5 reach receiver, 3 is retransmitted, When 4,5 ACKs reach sender, 3 reaches receiver). Total time = 20 ms (sending & receiving acks for 1, 2) + 20 ms (sending & receiving acks for 3, 4) + 10 ms for retransmitted packet 3's ack to reach sender = 50 ms. Note: Technically when 4,5 are sent back to sender, the last in-order ACK (2) is sent in the message, with hints that 4 and 5 have been received. This still informs the sender that 4, 5 have been received, so we can treat them as "acks" for the purposes of this question.

Flow Control - ACK Clock



Flow Control - Sliding Window Protocol (2)

- Flow control on receiver's side
  - In order to avoid loss caused by user application not calling recv(), receiver tells sender its available buffer space (WIN)
  - Sender uses lower of the WIN and W as the effective window size
- How to set a **timeout** for retransmission on sender's side?
  - Adaptively determine timeout value based on smoothed estimate of RTT

$$SRTT_{N+1} = 0.9*SRTT_N + 0.1*RTT_{N+1}$$
$$Svar_{N+1} = 0.9*Svar_N + 0.1*|RTT_{N+1} - SRTT_{N+1}|$$
$$\text{TCP Timeout}_N = SRTT_N + 4*Svar_N$$

Exercise: Adaptive Timeout

$$SRTT_{N+1} = 0.9*SRTT_N + 0.1*RTT_{N+1}$$
$$Svar_{N+1} = 0.9*Svar_N + 0.1*|RTT_{N+1} - SRTT_{N+1}|$$
$$\text{TCP Timeout}_N = SRTT_N + 4*Svar_N$$

A TCP connection is using adaptive timeout to detect packet loss. The initial SRTT is 20ms and SVar is 10ms.

1) How long should the TCP connection wait before timing out on the first packet (recall that TCP timeout uses 4 variances)?

2) Suppose the first packet was sent with an RTT of 20ms. What is the new timeout?

3) Suppose the second packet was sent with an RTT of 10 ms. What is the new timeout?

Exercise: Adaptive Timeout

$$SRTT_{N+1} = 0.9*SRTT_N + 0.1*RTT_{N+1}$$
$$Svar_{N+1} = 0.9*Svar_N + 0.1*|RTT_{N+1} - SRTT_{N+1}|$$
$$TCP\ Timeout_N = SRTT_N + 4*Svar_N$$

A TCP connection is using adaptive timeout to detect packet loss. The initial SRTT is 20ms and SVar is 10ms.

1) How long should the TCP connection wait before timing out on the first packet?

= 20 + 4 * 10 = 60 ms

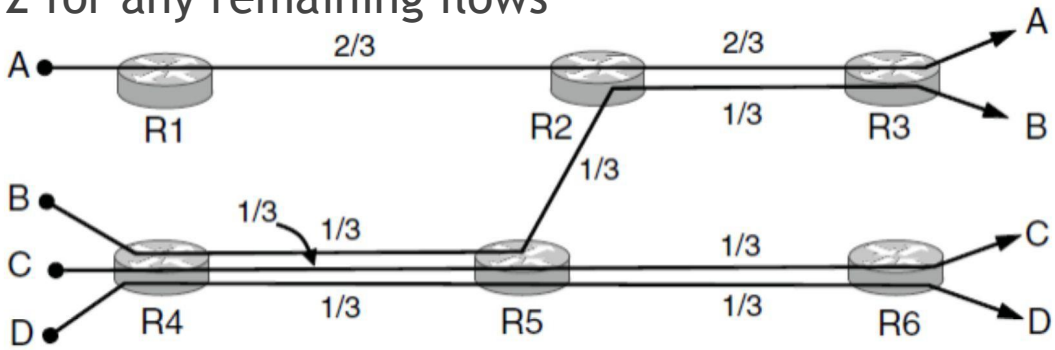
2) Suppose the first packet was sent with an RTT of 20ms. What is the new Timeout?

SRTT_new = 0.9 * 20 + 0.1 * 20 = 20
Svar = 0.9 * 10 + 0.1 * (20 - 20) = 9
New timeout = 20 + 4 * 9 = 56 ms

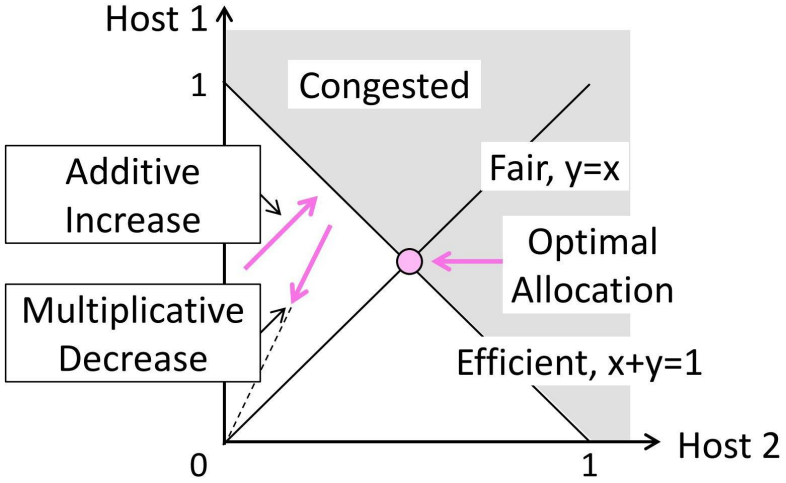
3) Suppose the second packet was sent with an RTT of 10 ms. What is the new timeout?
SRTT_new = 0.9 * 20 + 0.1 * 10 = 19
Svar = 0.9 * 9 + 0.1 * (10 - 1) = 9
New timeout = 19 + 4 * 9 = 55 ms

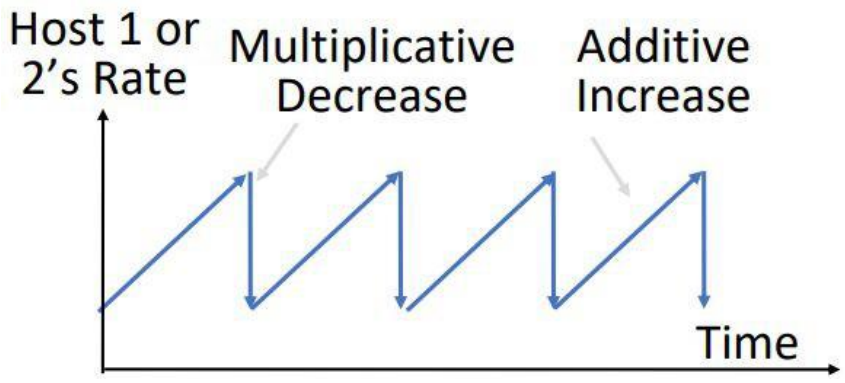
Max-Min Fair Allocation

1. Start with all flows at rate 0
2. Increase the flows until there is a new bottleneck in the network
3. Hold fixed the rate of the flows that are bottlenecked
4. Go to step 2 for any remaining flows



AIMD - Additive Increase Multiplicative Decrease

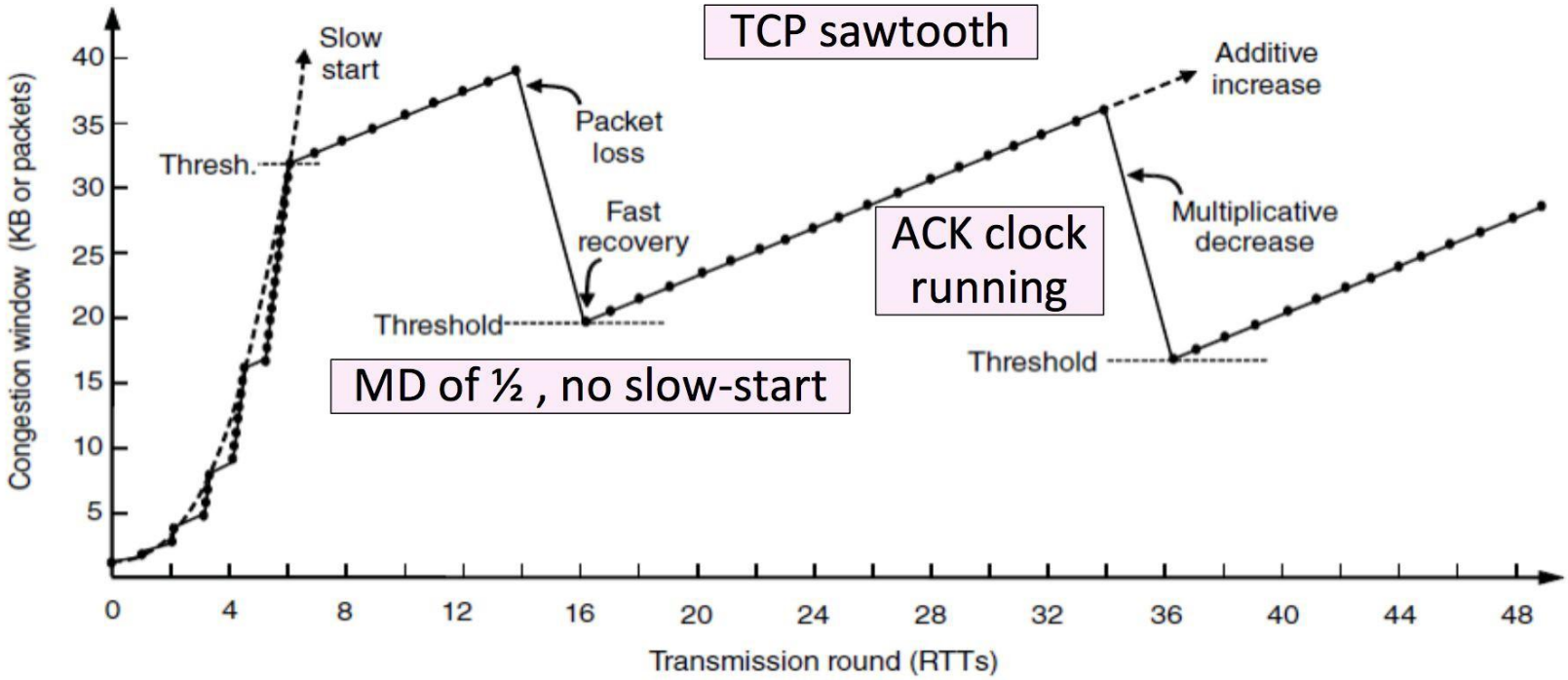




AIMD

- **Slow-Start** (used in AI)
  - Double cwnd until packet timeout
  - Restart and double until cwnd/2, then AI
- **Fast-Retransmit** (used in MD)
  - Three duplicate ACKs = packet loss
  - Don’t have to wait for TIMEOUT
- **Fast-Recovery** (used in MD)
  - MD after fast-retransmit
  - Then pretend further duplicate ACKs are the expected ACKs

TCP Reno


TCP sawtooth
Slow start
Thresh.
Packet loss
Fast recovery
Threshold
MD of ½ , no slow-start
Additive increase
ACK clock running
Multiplicative decrease
Threshold
Congestion window (KB or packets)
Transmission round (RTTs)
0
4
8
12
16
20
24
28
32
36
40
44
48
5
10
15
20
25
30
35
40

Network-Side Congestion Control

- Explicit Congestion Notification (**ECN**)
  - Router detects the onset of congestion via its queue. When congested, it marks affected packets in their IP headers
  - Marked packets arrive at receiver; treated as loss. TCP receiver reliably informs TCP sender of the congestion

Network Layer

- Connect different networks (send packets over multiple networks)
- Why do we need the network layer? Drawbacks of switches:
  - Don’t scale to large networks
  - Don’t work across more than one link layer technology
  - Don’t give much traffic control

IP Addresses Prefix and Forwarding (CIDR)

- **IP prefix a.b.c.d/L**
  - Represents addresses that have the same first L bits
  - e.g. 128.13.0.0/16 -> all 65536 addresses between 128.13.0.0 to 128.13.255.255
  - e.g. 18.31.0.0/32 -> 18.31.0.0 (only one address)
- **Longest Matching Prefix**
  - Find the longest prefix that contains the destination address, i.e., the most specific entry

Consider the following forwarding table with 2 rules. For a packet with destination address 192.24.29.32, what would its next hop be?

Prefix	Next Hop
192.24.0.0/18	D
192.24.12.0/22	B

```
Route to D: 192.00011000.00xxxxxx.xxxxxxxx
Route to B: 192.00011000.000011xx.xxxxxxxx
Packet:     192.00011000.00011101.00100000
```

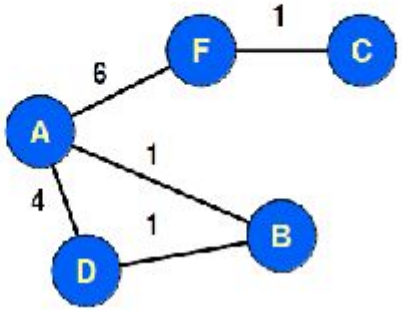
Overview of Intradomain Routing

- There are 2 main classes of routing approaches:
  - Distance Vector Routing
  - Link-State Routing
- Both approaches are distributed algorithms
  - No centralized authorities, no one can see the entire network from where they are
  - Nodes share network information with each other in a consistent manner
  - Nodes all update their information in the same way
  - Eventually nodes independently converge and have some view of the network
- Distance Vector Routing
  - Node X shares with **neighbors** the distance from Node X to everyone else
  - Node X eventually knows what hop to take to get to everyone else, with the lowest cost
- Link-State Routing
  - Node X shares with **everyone** Node X's distance to Node X's neighbors
  - Node X eventually has a global view of the network and can calculate the best paths

Distance Vector Routing

- Distance Vector Routing
  - Node X shares with **neighbors** the distance from Node X to everyone else
  - Node X eventually knows what hop to take to get to everyone else, with the lowest cost
- A broad class of routing methods that involving sharing “distance vectors”
  - (i.e. an array of numbers representing its distance to other nodes)
  - Everyone shares distance vectors and uses this to update their own distance vectors (by using the lowest cost path)
- Distributed version of Bellman-Ford algorithm
  - “Simply put, the algorithm initializes the distance to the source to 0 and all other nodes to infinity. Then for all edges, if the distance to the destination can be shortened by taking the edge, the distance is updated to the new lower value.”
- Example of a distance vector protocol:
  - Routing Information Protocol (RIP)

Consider the network below.



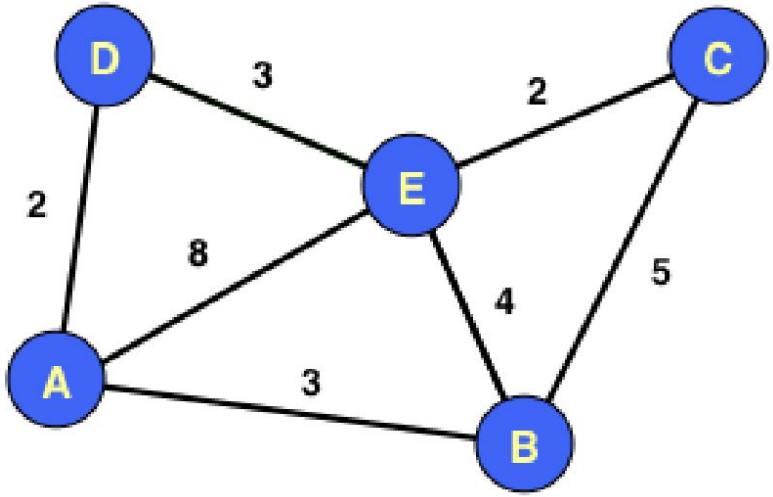
a) Show the initial table for A for the Distance Vector//Bellman-Ford method of shortest path calculation.

b) Show the final shortest path table for A.

Dest	Cst	Next Hop
A	0	A
B	1	B
C	7	F
D	2	B
F	6	F

Link-State Routing

- Link-State Routing
  - Node X shares with **everyone** Node X's distance to Node X's neighbors
  - Node X eventually has a global view of the network and can calculate the best paths
- A broad class of routing methods that involving sharing "link-state"
  - (i.e. the current state of a node's links with its direct neighbors)
  - Everyone floods everyone else's link-state, until everyone knows everyone's link-state
- Once a node has a global view, they use Dijkstra's algorithm
  - Can easily calculate the shortest path from one node to all other nodes
- Examples of link-state routing protocols:
  - Intermediate System to Intermediate System (IS-IS)
  - Open Shortest Path First (OSPF)



Derive the shortest path using Dijkstra's algorithm. Show each step of the algorithm by filling in the table below.

Iteration	Finalized Nodes List	Total Cost from A				
		Node A	Node B	Node C	Node D	Node E
0	{A}	0	3	∞	2	8
1	{A, D}	0	3	∞	2	5
2	{A, D, B}	0	3	8	2	5
3	{A, D, B, E}	0	3	7	2	5
4	{A, D, B, E, C}	0	3	7	2	5