THE UNIVERSITY OF
MELBOURNE

Week 10
Transport Layer

COMP90007 Internet Technology

Prepared by: Chenyang Lu (Luke)

Your Tutor

Chenyang Lu (Luke)

- Email: chenyang.lu@unimelb.edu.au

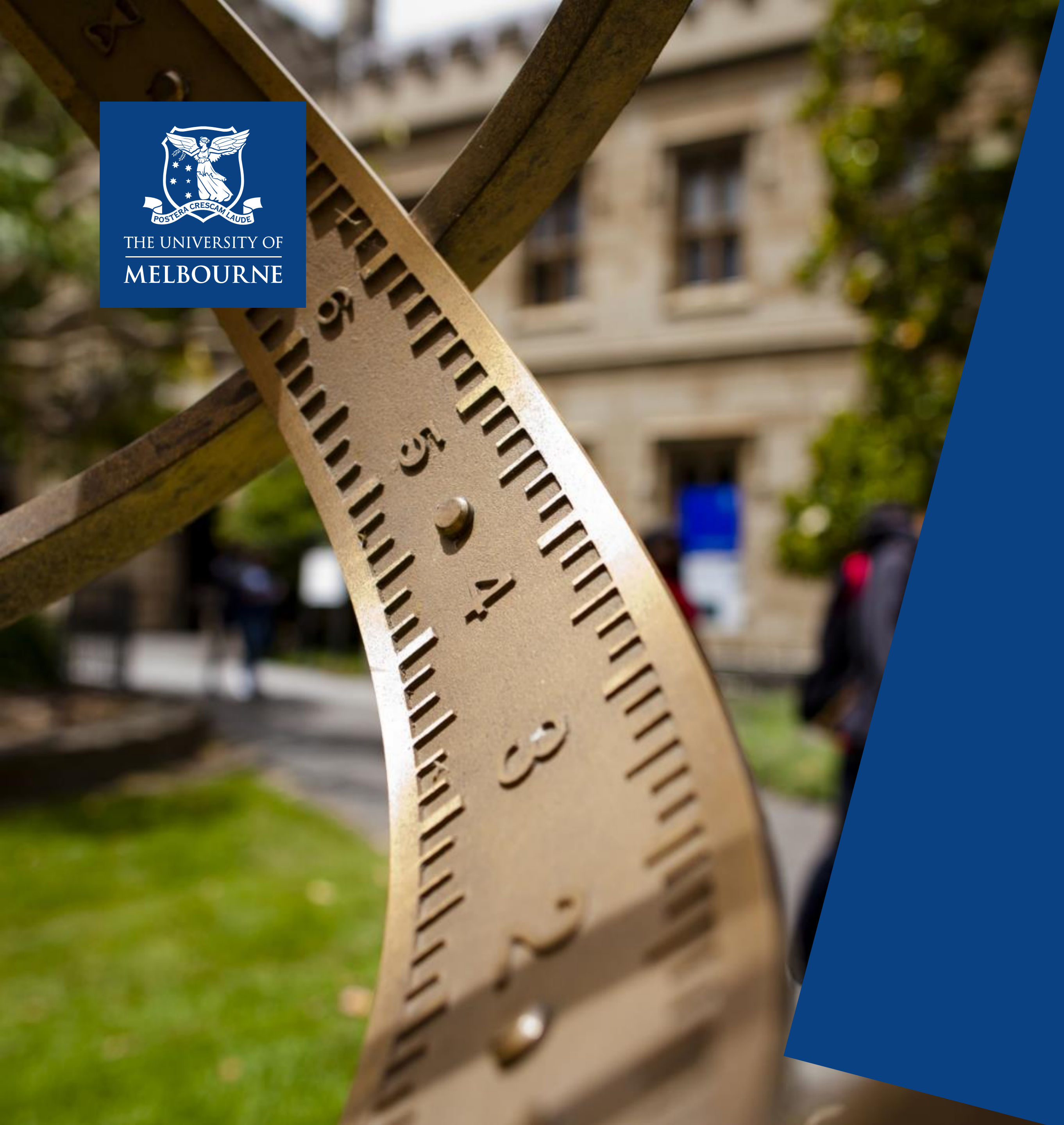
- Workshop Slides: https://github.com/LuChenyang3842/Internet-technology-teaching-material

Day	Time	Location
Tue	18:15	Bouverie st –B114
Wed	10:00	Elec Engineering -122
Wed	17:15	Bouverie-sr 132

Tutor Feedback

https://apps.eng.unimelb.edu.au/casmas/index.php?r=qoct/feedback&subjCode=COMP90007

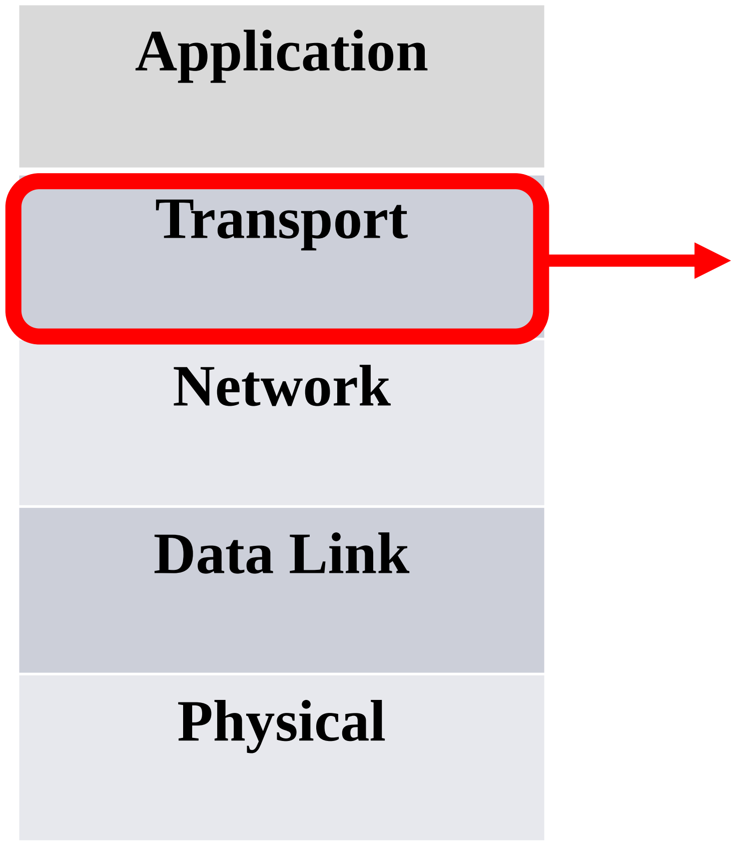
For students who attend Wednesday 10:00 am Tutorial, please choose Wed 5:15pm in dropdown list to leave feedback.



THE UNIVERSITY OF MELBOURNE

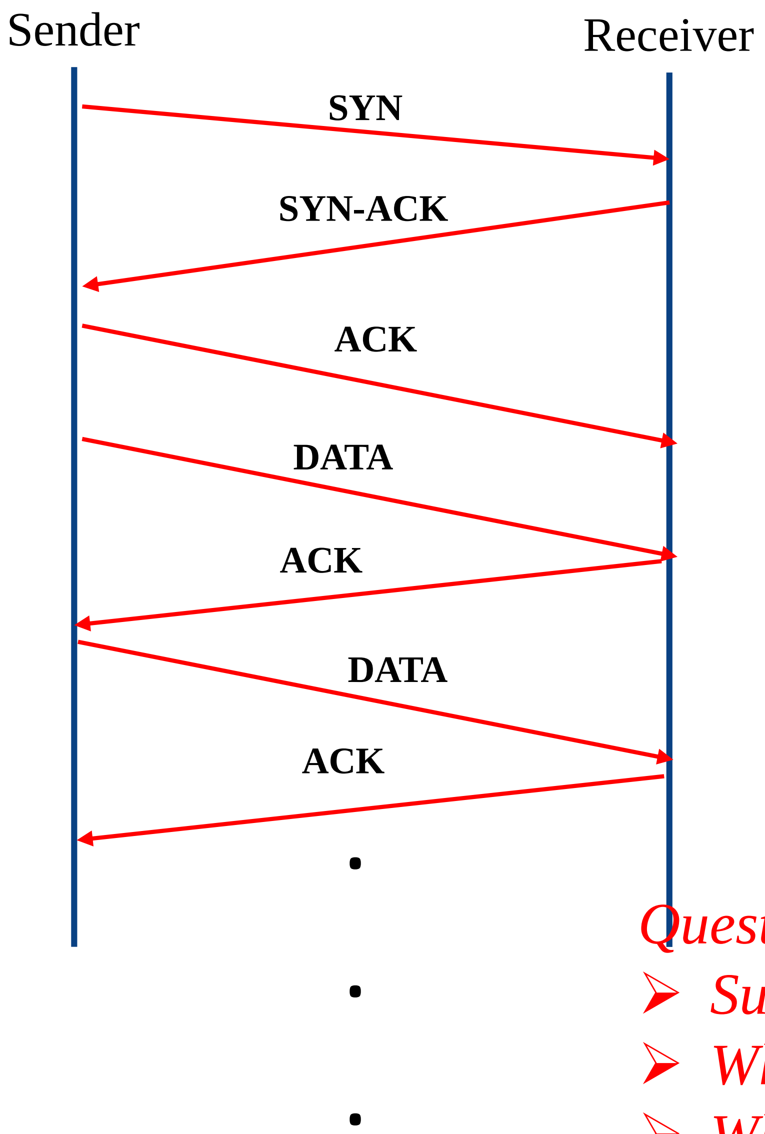
Transport Layer

Transport Layer

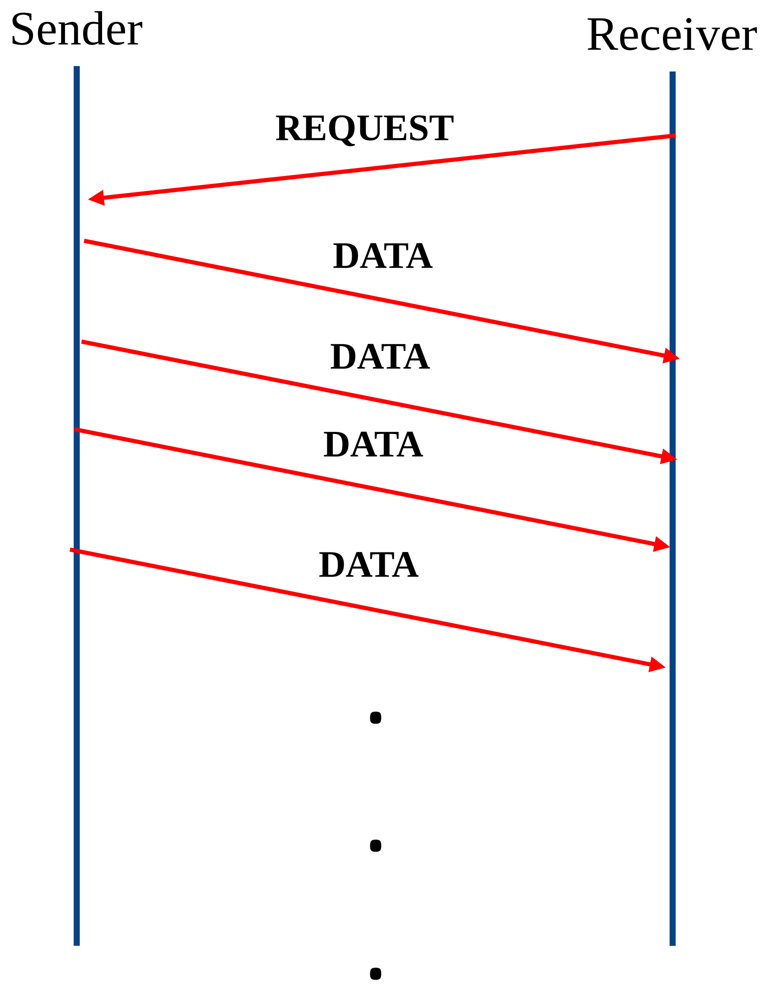


1. Transport Layer Encapsulation
2. Transmission Control Protocol (TCP)
   - Connection-establishment
     - Three-way handshake
   - Connection-release
     - Asymmetric
     - Symmetric
   - TCP segment header
   - TCP based socket
   - Error-control
   - Flow-control
   - Congestion-control
     - ❑ Slow start, Additive increase
     - ❑ Tahoe, Reno
3. User Datagram Protocol (UDP)
4. Techniques for achieving Quality of Service (QoS)

TCP



UDP



Question:

- *Summarize the differences between them?*
- *Which one is more reliable, why?*
- *Which one is faster, why?*

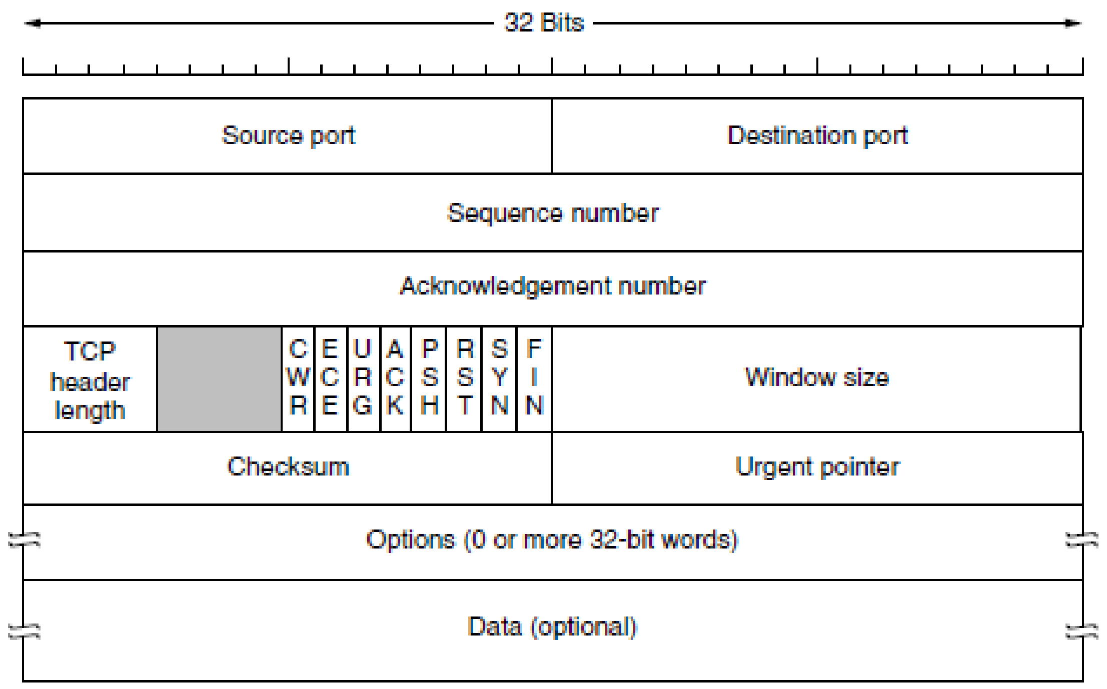
TCP

- Connection-oriented
  - <u>Establish connection</u> before transmission
  - <u>Connection release</u> after transmission
  - Sender will wait for ack before sending the next segment
- More Reliable
  - If lost of data, will send again
  - Discard duplicated segment
  - Recipient will receive data in order

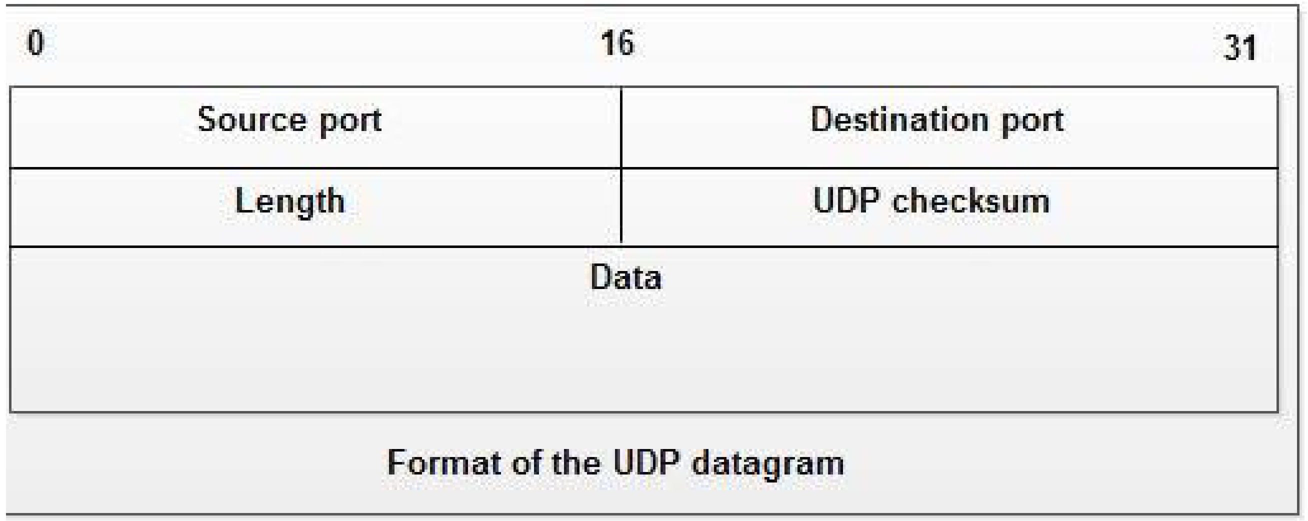
UDP

- Connectionless
  - No connection required before transmission
  - The sender will not wait to make sure the recipient received the segment
- Faster
  - Less over head, no wait for acknowledgement
  - UDP is used when speed is desirable and error correction is not necessary (e.g. live streaming, online game)

TCP segment header



UDP segment header



Question 1

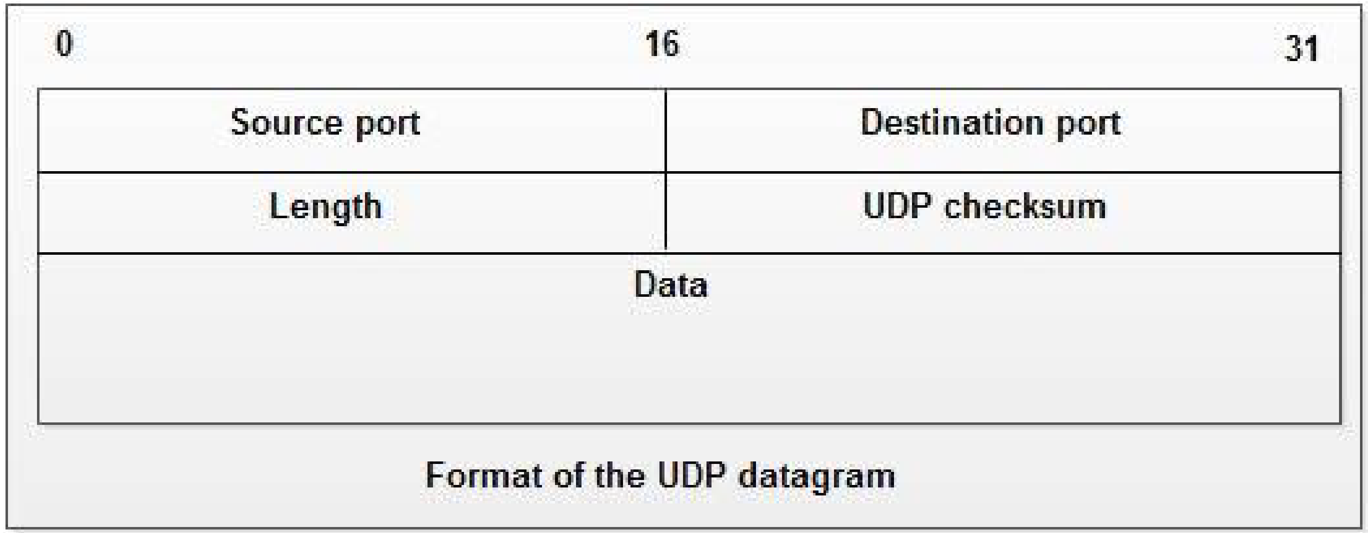
Why does UDP exist? Would it not have been enough to just let the user processes send raw IP packets?

Answer:

No.

- IP packets contain IP addresses, which specify a destination machine.
- Once such a packet arrived, how would the network handler know which process to give it to?
- UDP packets contain a **destination port**. This information is essential so they can be delivered to the correct process.

UDP segment header



Question 2

Both UDP and TCP use port numbers to identify the destination entity when delivering a message.
Discuss possible reasons for why these protocols invented a new **abstract ID (port numbers),** instead of using **process IDs** (which already existed when these protocols were designed?)

Answer:
Here are three reasons.

- First, process IDs are OS-specific. Using process IDs would have made these protocols OS-dependent.
- Second, a single process may establish multiple channels of communications. A single process ID (per process) as the destination identifier cannot be used to distinguish between these channels.
- Third, having processes listen on well known ports is easy, but well-known process IDs are impossible.

TCP - Congestion Control

Question:

- *What are the two factors that determining maximum window size (i.e. maximum transmission rate)?*

TCP - Congestion Control

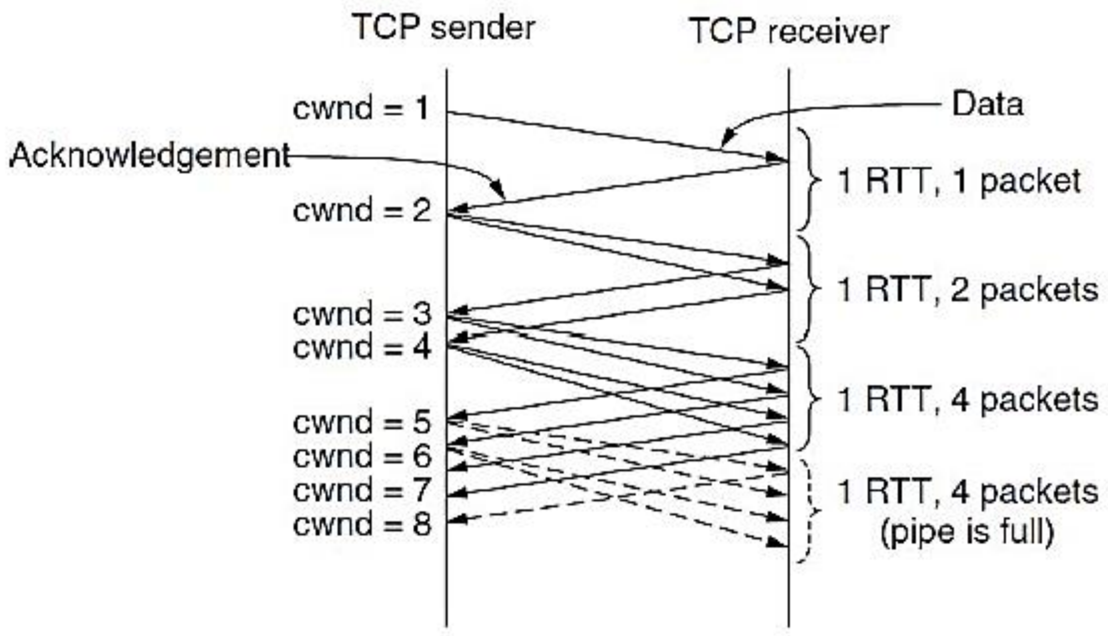
The sender maintains two windows actually

- Window described by the receiver (receiver capacity)
- Congestion window (network capacity)

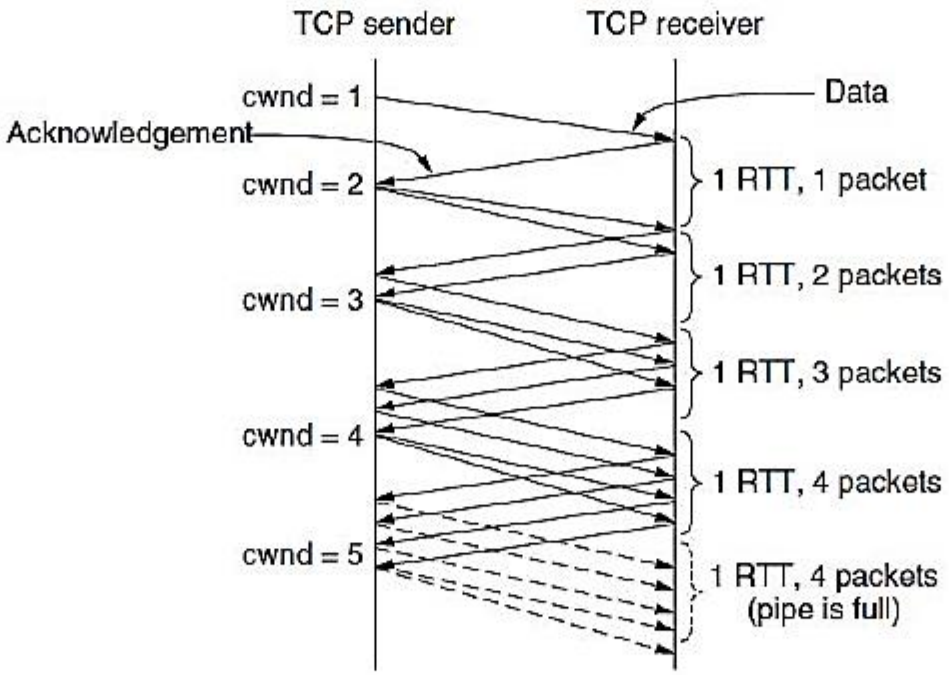
Each regulates the number of bytes the sender can transmit – the maximum transmission rate is the **minimum of the two windows**

TCP - Congestion Control

Slow Start



Addictive increase

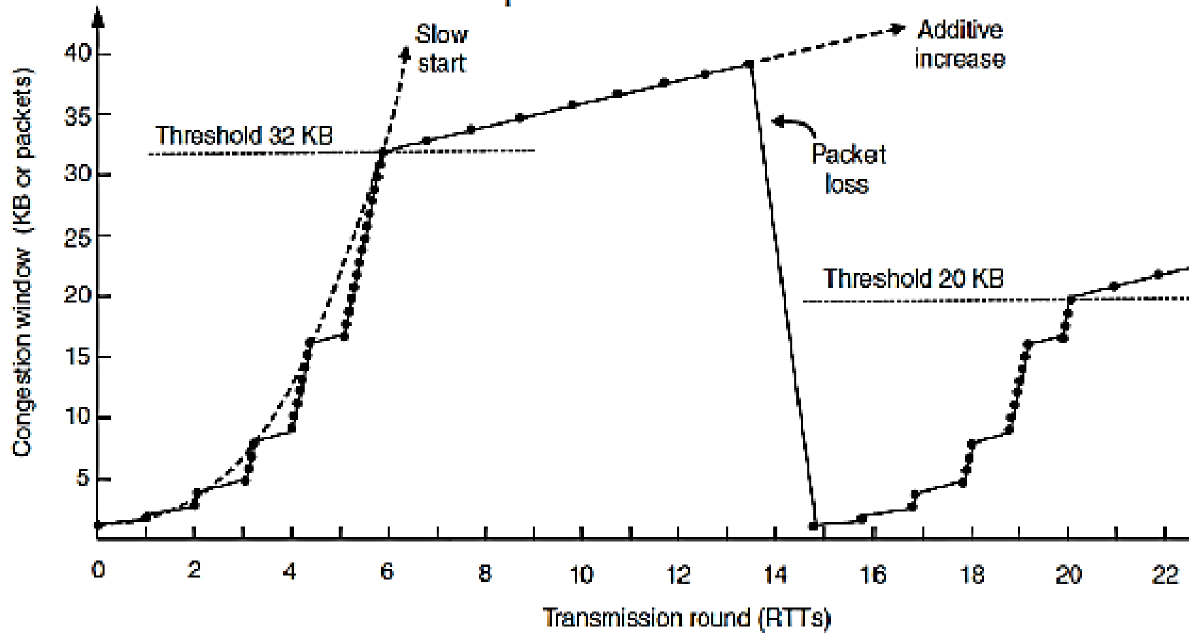


Question 3 Congestion Control

What is the key difference between TCP Tahoe and TCP Reno?

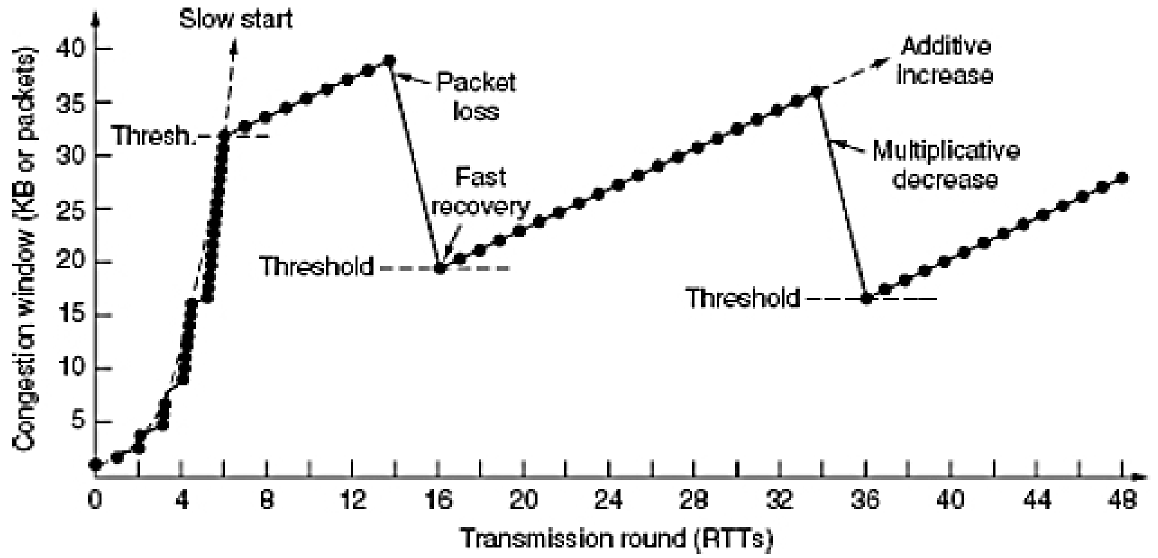
Tahoe

Slow start followed by additive increase (TCP Tahoe)
Threshold is half of previous

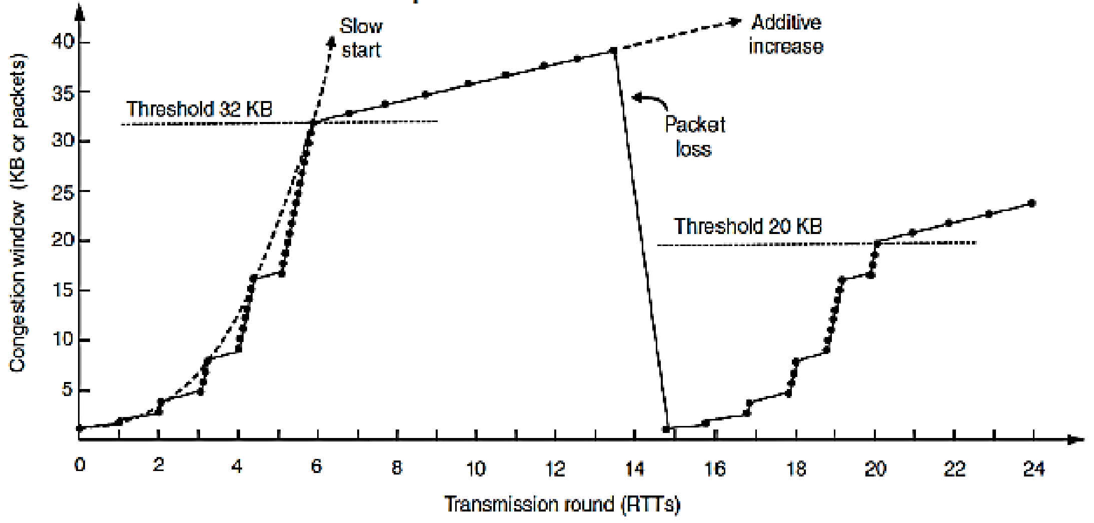


Reno

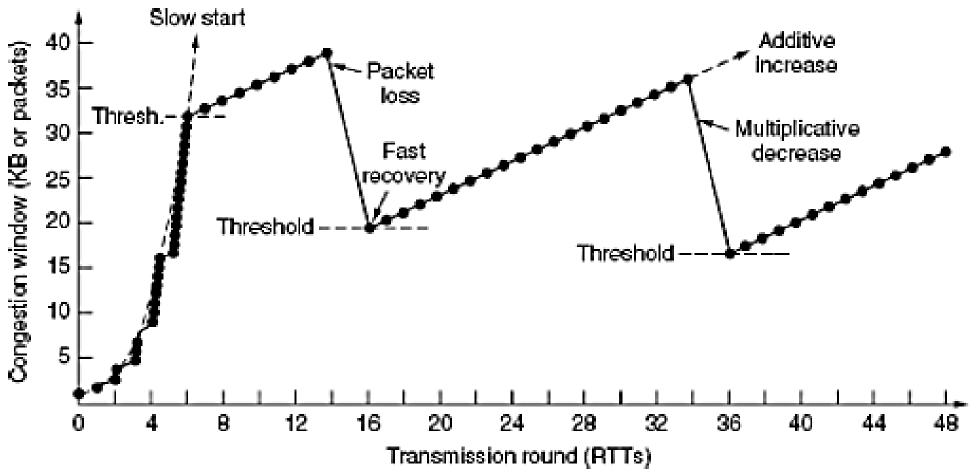
Another one with TCP Reno



Slow start followed by additive increase (TCP Tahoe)
Threshold is half of previous



Another one with TCP Reno



Answer:

Tahoe: *Slow start (from 1 segment), Whenever the slow start threshold is crossed, TCP switches from slow start to additive increase. Once congestion windows full, slow start again.*

Reno: *Slow start (from 1 segment), Whenever the slow start threshold is crossed, TCP switches from slow start to additive increase, once congestion windows full, Skipping slow start and going directly to additive increase. The overall algorithm here is called fast recovery.*

The key difference, and benefit of Reno, is that Reno avoids Slow start when it can and can do Fast recovery at certain cases.

Techniques for Achieving Quality of Service (QoS)

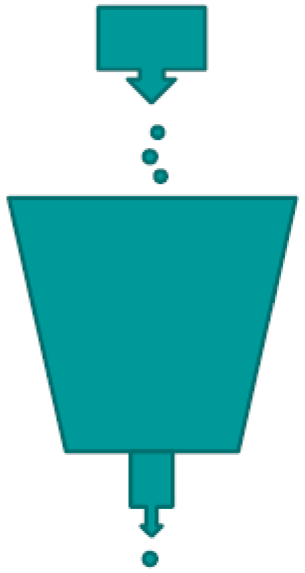
- **Over-provisioning**
  - more than adequate buffer, router CPU, and bandwidth (expensive and not scalable ...)

- **Buffering**
  - Buffer received flows before delivery - increases delay, but smoothes out jitter, no effect in reliability or bandwidth

- **Traffic Shaping**
  - regulate the average rate of transmission and burstiness of transmission
  - **leaky bucket**
  - **token bucket**

And more…. Check lecture slides

Leaky Bucket



Large **bursts** of traffic is buffered and smoothed while sending

E.g. can be done at host sending data

Question 4

Recall the **Leaky Bucket algorithm** we saw in class.
If a sender has **a burst data rate of 20KB/s for 20 seconds** as data to send and we have **a bucket size of 100KB** with an **output rate of 10KB/s:**
Does the Leaky Bucket algorithm achieve its aim of regulating output properly? If so explain how and show your calculations. If not, find the **appropriate bucket size** and discuss why we need more/less of a bucket size.

Answer:

- The bucket is too small, we should have had a bucket size of 200KB.
- The input data is 20x20=400KB,
- In the same time the bucket can empty only 20x10=200KB and can hold another 100KB.
- So another 100KB bucket size is needed to deal with 400KB in total.

Question 5

TCP relies on timers for resending in case some ACKs are missing. If **we set such timers to a fixed value of say 100ms,** discuss what would be the **advantages** and **disadvantages** of such a static protocol design.

Answer: A fixed 100ms timer is easy to implement and does not require monitoring network status and load. **If conditions are stable** and 100ms is set accordingly then all should work fine with little overhead.

However, in most cases, network conditions change.

- When network is busy... Thus 100ms could lead to prematurely timers going off, which means many redundant resends,
- When network conditions indicate fast traffic, then the opposite would be true, i.e., static timers will go off too late for missing segments and applications would wait extra for no good reason.

In case of dynamic conditions in the network, timers should be set based on measurements rather than to a static value.