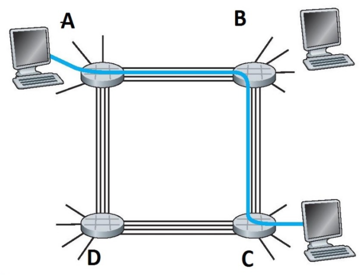
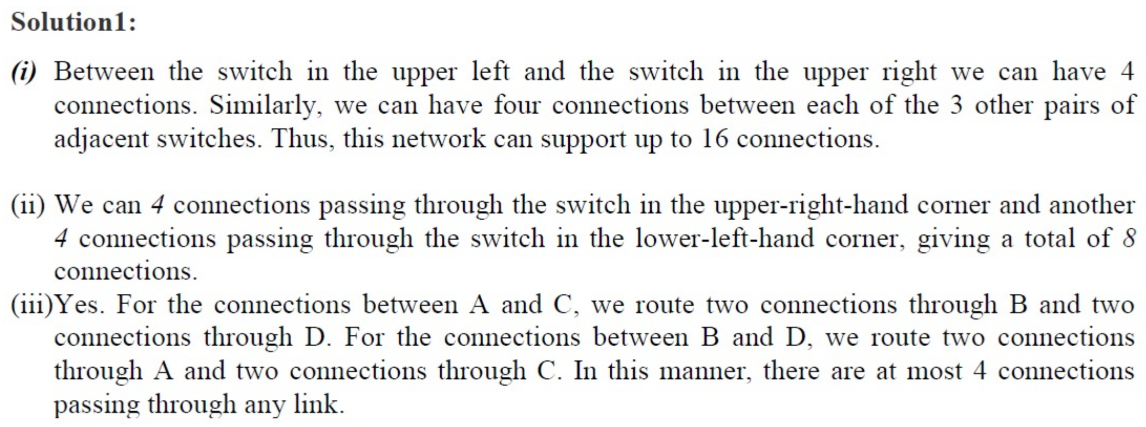
Lecture 11

Problem 11:

- Consider the following circuit-switched network in Fig 1, and four switches A, B, C and D, going in the clockwise direction, and 4 circuits on each link.
- (i) In this network, calculate the **maximum number** of connections that can be in progress **simultaneously** at any one time?
- (ii) If all connections are between switches A and C, what is the **maximum number** of connections that can be in progress simultaneously?
- (iii) If we want to have **4 connections** between A and C, and another **4 connections** between B and D, can we route these calls through the 4 links to accommodate all 8 connections?





Solution1:

(i) Between the switch in the upper left and the switch in the upper right we can have 4 connections. Similarly, we can have four connections between each of the 3 other pairs of adjacent switches. Thus, this network can support up to 16 connections.

(ii) We can *4* connections passing through the switch in the upper-right-hand corner and another *4* connections passing through the switch in the lower-left-hand corner, giving a total of *8* connections.

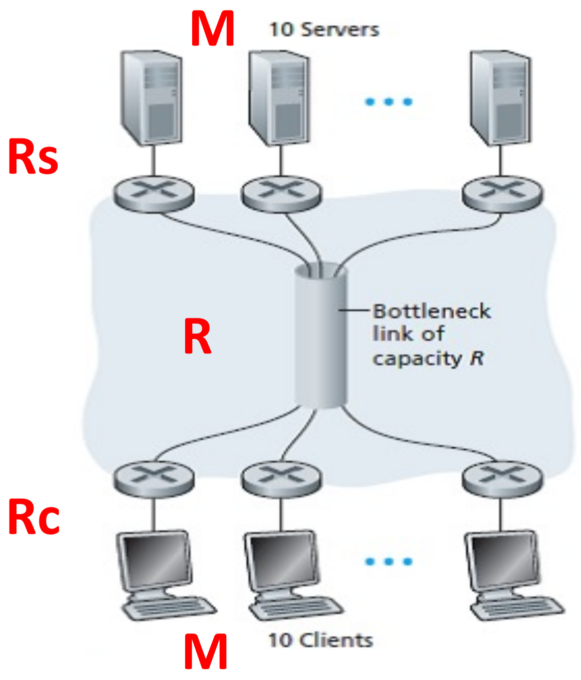
(iii)Yes. For the connections between A and C, we route two connections through B and two connections through D. For the connections between B and D, we route two connections through A and two connections through C. In this manner, there are at most 4 connections passing through any link.

Problem 12:

Solution2:

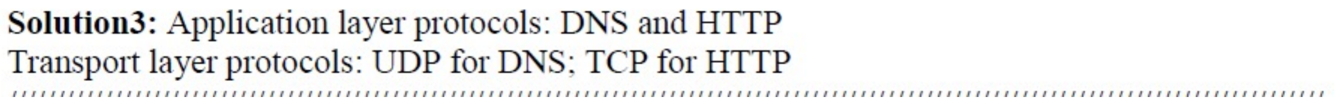
Throughput = $min\{R_s, R_c, R/M\}$

- Consider the throughput example corresponding to Fig.2.
- If we have are M client-server pairs instead of 10.
- Denote Rs, Rc, and R for the rates of the server links, client links, and network link.
- If all other links have enough capacity and that there is no other traffic in the network besides the traffic generated by the M client-server pairs.
- Please derive a general expression for throughput in terms of Rs , Rc , R , and M .



Problem 13:

- Assume an HTTP client that wants to obtain a web document at a given URL.
- The IP address of the HTTP server is unknown at first.
- The web document at the URL has one embedded GIF image that resides at the same server as the original document.
- What transport and application layer protocols besides HTTP are needed in this scenario?


Solution3: Application layer protocols: DNS and HTTP
Transport layer protocols: UDP for DNS; TCP for HTTP

Problem 14

- Suppose you click on a link to obtain a Web page in your Web browser.
- The IP address for the associated URL is not cached in your local host, so a DNS lookup is necessary to obtain this IP address.
- Assume that **n** DNS servers are visited before your host receives the IP address from DNS; the successive visits incur an RTT of RTT1,...,RTTn.
- Further assume that the Web page associated with the link contains exactly **one object**, consisting of a small amount of HTML text.
- Let **RTT0** denote the RTT between the local host and the server containing this object.
- Assuming zero transmission time of the object, how much time it will take from when the client clicks on the link until the client receives the object?

Solution 4: The total amount of time to get the IP address is
$RTT_1 + RTT_2 + \cdots + RTT_n$.
Once the IP address is known, RTT_O elapses to set up the TCP connection and another RTT_O elapses to request and receive the small object. The total response time is
$2RTT_o + RTT_1 + RTT_2 + \cdots + RTT_n$

Problem 15:

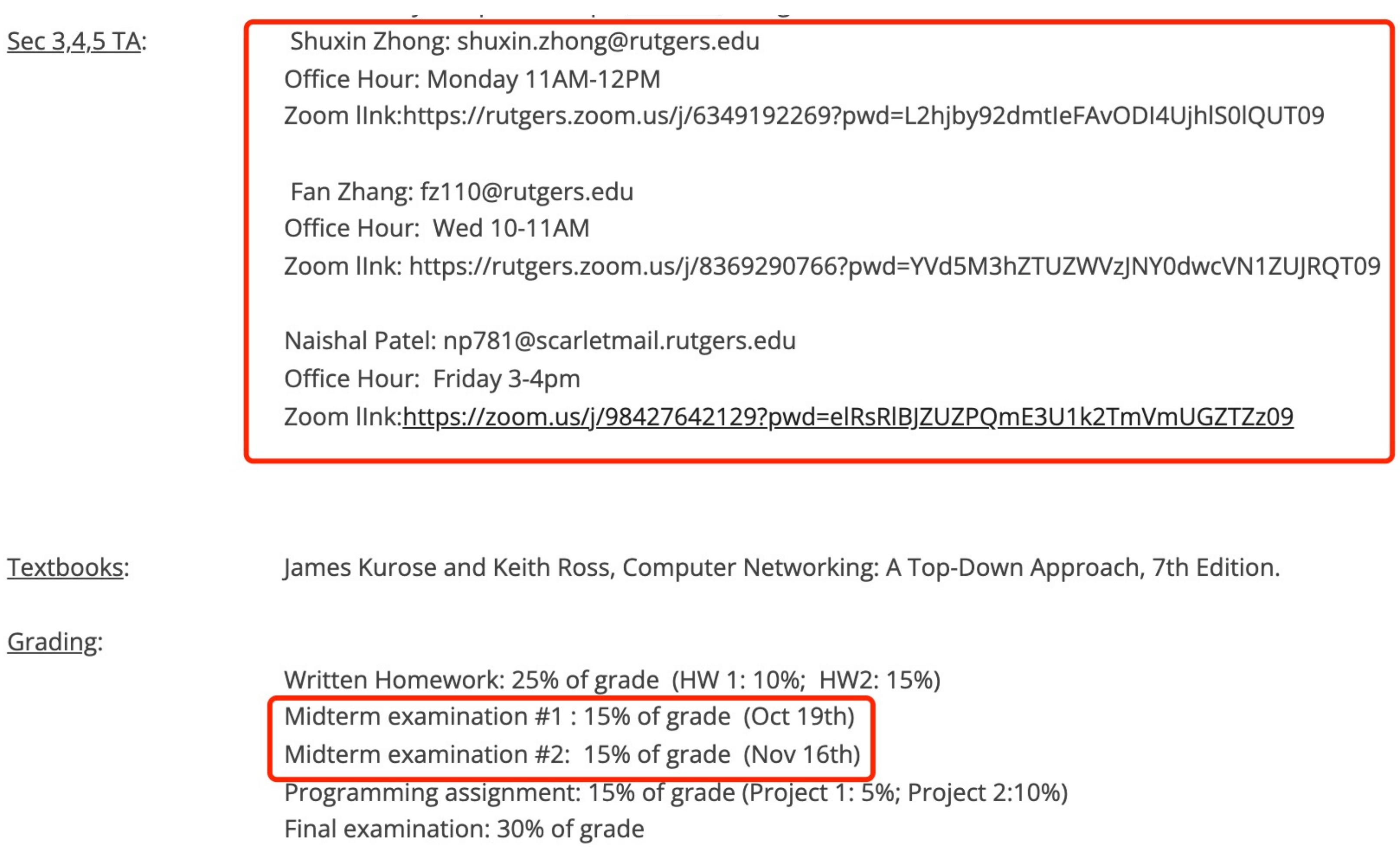
- Referring to the above problem, suppose this HTML file references 8 very small objects on the same server. Neglecting transmission times, how much time elapses with
- (i) Non-persistent HTTP with no parallel TCP connections?
- (ii) Non-persistent HTTP with the browser configured for 5 parallel connections?
- (iii) Persistent HTTP?

(i) $RTT_1 + \cdots + RTT_n + 2RTT_o + 8 \cdot 2RTT_o = 18RTT_o + RTT_1 + \cdots + RTT_n$.

(ii) $RTT_1 + \cdots + RTT_n + 2RTT_o + 2 \cdot 2RTT_o = 6RTT_o + RTT_1 + \cdots + RTT_n$

(iii) $RTT_1 + \cdots + RTT_n + 2RTT_o + 8RTT_o$
$= 10RTT_o + RTT_1 + \cdots + RTT_n$.

TAs, Their Office Hours and Midterms



Sec 3,4,5 TA:

Shuxin Zhong: shuxin.zhong@rutgers.edu
Office Hour: Monday 11AM-12PM
Zoom lInk:https://rutgers.zoom.us/j/6349192269?pwd=L2hjby92dmtIeFAvODI4UjhlS0lQUT09

Fan Zhang: fz110@rutgers.edu
Office Hour: Wed 10-11AM
Zoom lInk: https://rutgers.zoom.us/j/8369290766?pwd=YVd5M3hZTUZWVzJNY0dwcVN1ZUJRQT09

Naishal Patel: np781@scarletmail.rutgers.edu
Office Hour: Friday 3-4pm
Zoom lInk:https://zoom.us/j/98427642129?pwd=elRsRlBJZUZPQmE3U1k2TmVmUGZTZz09

Textbooks: James Kurose and Keith Ross, Computer Networking: A Top-Down Approach, 7th Edition.

Grading:

Written Homework: 25% of grade (HW 1: 10%; HW2: 15%)
Midterm examination #1 : 15% of grade (Oct 19th)
Midterm examination #2: 15% of grade (Nov 16th)
Programming assignment: 15% of grade (Project 1: 5%; Project 2:10%)
Final examination: 30% of grade

Two Projects

No Team, e.g., every one submit his/her own code.

Project 1 (5%): Release Oct 10 and Due Oct 27th

Project 2 (10%): Release Oct 29th and Due Nov 24th

TA: Naishal Patel:

np781@scarletmail.rutgers.edu
Office Hour: Friday 3-4pm
Zoom lInk:
https://zoom.us/j/98427642129?pwd=elRsRlBJZUZPQmE3U1k2TmVmUGZTZz09

This project 1 is to let you explore the socket programming interface in and some other basic aspects of the Python language.

This exercise serves as the foundation for the upcoming programming project.

A sample working code is given to you in proj0.py.

The program consists of server code and client code written as two separate threads.

(1) Understand the functionality implemented in the program. First, download, save and execute the program as is in your environment. Make sure it executes successfully and according to how you would expect.

Then attempt the changes suggested below. It will help subsequent projects if you try playing around with the program as follows.

(2) Try running the program immediately again when it finishes successfully. What do you see? Why? What happens when you remove the various sleep()s in the program?

(3) Separate the server code and client code into two different programs, server.py and client.py. Execute the server program first and then execute the client program. You should still get the same set of print messages as in the combined threaded code (proj0.py)

(4) In the given code, the server just sends a message string to the client after it connects. Modify the program so that the client sends a string to the server and then the server reverses the string and sends it back to the client with the original string. For example, if the client sends "HELLO" to the server, the client should receive "OLLEHHELLO".
Your program should print the string sent by the client and the corresponding string received by the client. Your client program should be able to read a string (each line is a string) from a test file (say, in-proj0.txt). Similarly, the output of your program should be written to a file (say, out-proj0.txt).