Name_____ Student ID_____ Department/Year_____

3rd Examination

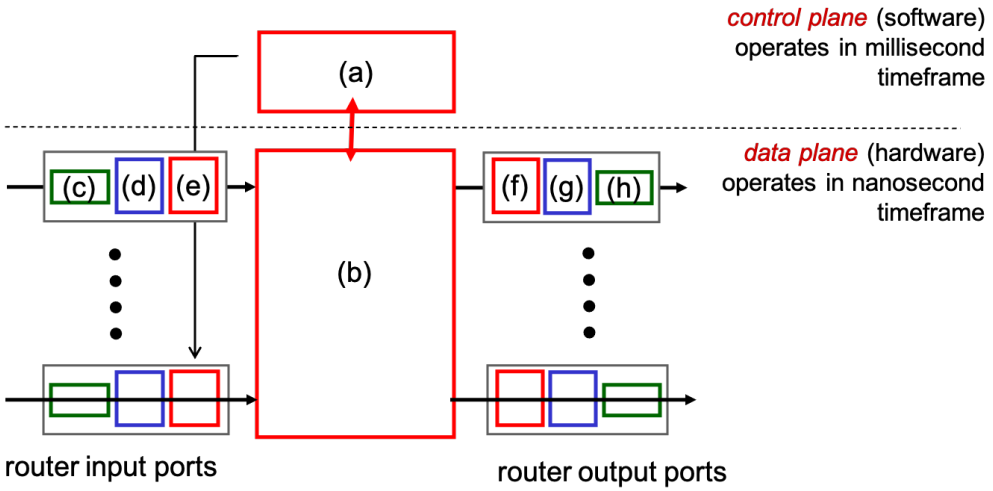
Introduction to Computer Networks (Online)
Class#: EE 4020, Class-ID: 901E31110
Fall 2021

10:20-12:10 Thursday
January 06, 2022

Cautions

1. There are in total 100 points to earn. You have 90 minutes to answer the questions. Skim through all questions and start from the questions you are more confident with.
2. Use only English to answer the questions. Misspelling and grammar errors will be tolerated, but you want to make sure with these errors your answers will still make sense.

1. (ch42, 7pt) Shown below is the high-level view of a generic router. See if you can tell the component implementing each of the router functions.



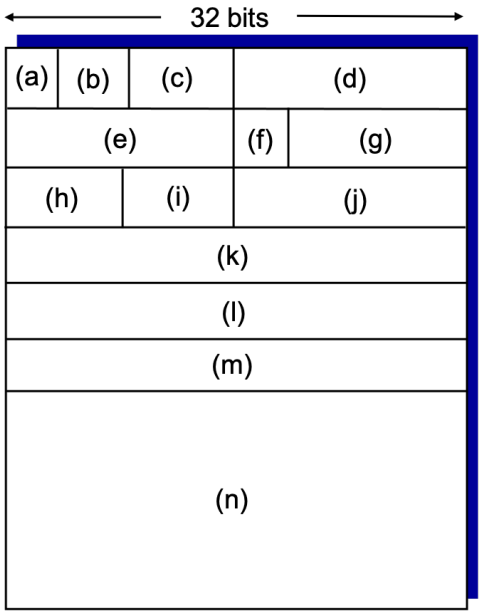
(1) Which component implements the route computation function (1pt)?
(2) Which component implements the packet switching function (1pt)?
(3) Which component implements the forwarding table lookup function (1pt)?
(4) Which component implements the dropping policy (1pt)?
(5) Which component implements the scheduling policy (1pt)?
(6) Which component implements the link layer function (1pt)?
(7) Which component implements the physical layer function (1pt)?

Sample Solution:

(1) (a)
(2) (b)
(3) (e)
(4) (e)(f)
(5) (e)(f)
(6) (d)(g)
(7) (c)(h)

(Grading policy: -1pt for any false addition/missing pick)

2. (ch43, 11pt) IPv4's packet format is quite complex. However, each field in the packet serves a purpose. Recall what each field is for and see if you can identify which field(s) is for each purpose below.



(1) Which field(s) tells the IP version? (1pt)
(2) Which field(s) tells whether the header is error free? (1pt)
(3) Which field(s) tells whether the packet is error free? (1pt)
(4) Which field(s) is looked at for traditional IP forwarding? (1pt)
(5) Which field(s) tells the offset of the 1st data byte in a packet? (1pt)
(6) If (m) field is empty, which field(s) is the minimum to tell the number of data bytes in a packet? (1pt)
(7) If (m) field is not empty, which field(s) is the minimum to tell the number of data bytes in a packet? (2pt)
(8) If (m) field is not empty, which field(s) are necessary to perform the IP fragmentation and assembly function (3pt)?

Sample Solution:

(1) (a)
(2) (j)
(3) none of the above
(4) (l)
(5) (b)
(6) (d)

(7) (b)(d)
(8) (b)(d)(e)(f)(g)

(Grading policy: -1pt for any false addition/missing pick)

3. (ch43, 5pt) Which of the following are the benefits of hierarchical addressing?
   (1) Smaller forwarding table. (1pt)
   (2) Faster forwarding table lookup. (1pt)
   (3) More flexible allocation of IP addresses to smaller organizations in an institute. (1pt)
   (4) Aggregation of route advertisement. (1pt)
   (5) Need of acquiring the netmask in DHCP. (1pt)

Sample Solution:

T, T, T, T, F

4. (ch43, 9pt) DHCP is the protocol commonly used to assigns IP addresses to machines that come and go dynamically. Recall the design of DHCP and argue for the design choices.
   (1) What information is assigned after a successful DHCP request? (3pt)
   (2) Which layer does DHCP operate on? (1pt)
   (3) Continue from (2), why? (2pt)
   (4) Which transport layer protocol is used to transmit DHCP packets? (1pt)
   (5) Continue from (4), why? (2pt)

Sample Solution:

(1) client IP address (lease, transaction ID), subnet mask, DNS server IP address, 1st-hop router IP address, domain name

(2) application layer

(3) complexity at the edge

(4) UDP

(5) any reasons that make sense. For example,

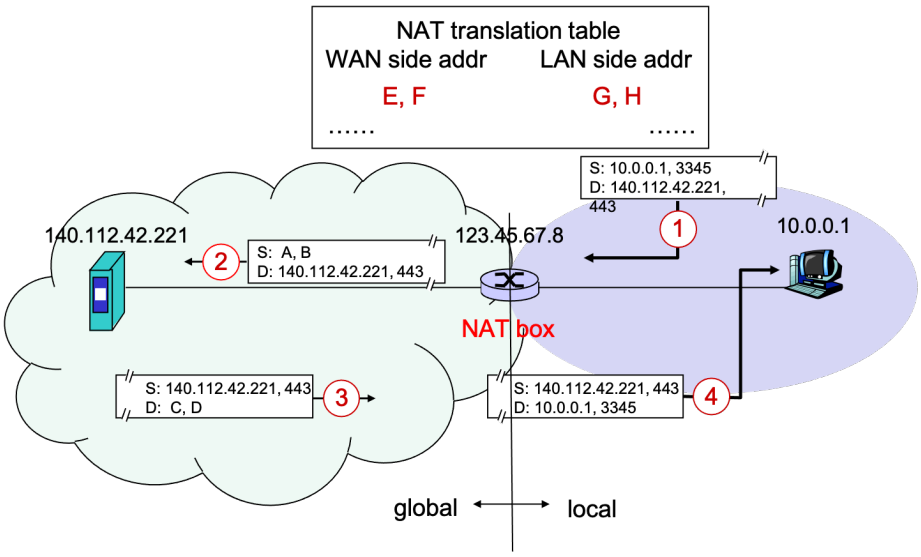
-- There's no IP address to set up a TCP connection at the first place.

-- DHCP are for dynamic uses. Shorter delay will better serve the purpose.

(Grading policy for (1): -1pt for any false addition/missing pick)

5. (ch43, 8pt) NAT allows a local network of multiple machines to share one IP address. What the protocol does is essentially to (1) translate the source address from local to global for outgoing packets and (2) translate the destination address from global to local for the returning packets.

   Depicted below is a scenario of a local machine (10.0.0.1) behind NAT (123.45.67.8) trying to request a page from a Web server (140.112.42.221). According to the protocol, the source address of the HTTPS request packet should be translated from local to global as it exits the local network via the NAT box. For the purpose, the NAT box assigns port 613 to allow the HTTPS reply packet to come back.



   (1) Tell the value of A and B in the HTTPS request. (2pt)
   (2) Tell the value of C and D in the HTTPS reply. (2pt)
   (3) Tell the value of E, F, G, H in the new WAN-LAN address mapping created. (2pt)
   (4) There are usually many mappings in the NAT box. How does the NAT box tell for a particular incoming packet which mapping to use to translate the destination address back? (2pt)

Sample Solution:

(1) 123.45.67.8, 613

(2) 123.45.67.8, 613

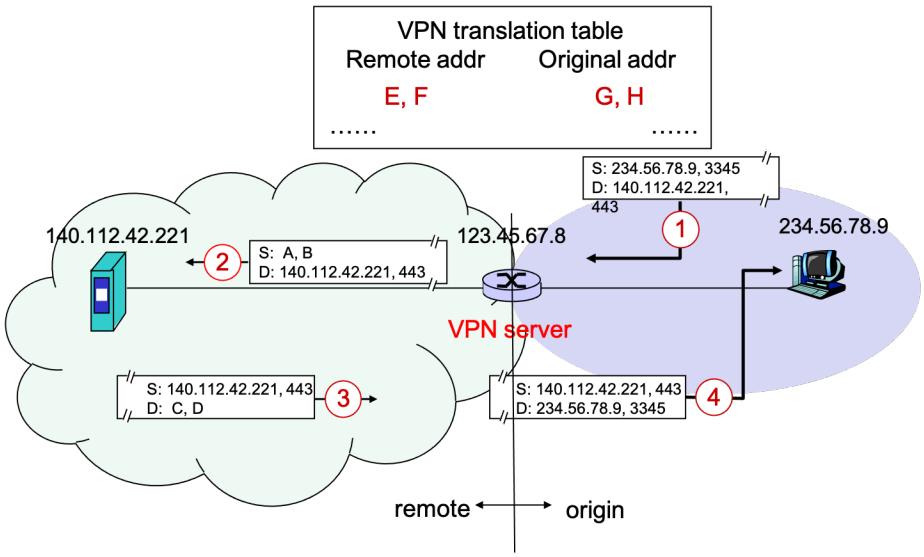
(3) 123.45.67.8, 613 – 10.0.0.1, 3345

(4) Match the destination port number of the incoming packet (D) to the WAN side address port number (F)

(Grading policy for (1)-(3): -1pt for any false/missing address or port value)

6. (ch44, 8pt) VPN allows a machine to share the IP address of a remote machine to get around firewalls that restrict access based on the source IP address. What the protocol does is essentially to (1) translate the source address from the origin machine to the remote machine for outgoing packets and (2) translate the destination address from the remote machine back to the origin machine for the returning packets.

   Depicted below is a scenario of the origin machine (234.56.78.9) connected to a VPN server (123.45.67.8) trying to request a page from a Web server (140.112.42.221). According to the protocol, the source address of the HTTPS request packet should be translated from the original address to the VPN server's address. For the purpose, the VPN server assigns port 613 to allow the HTTPS reply packet to come back.



   To this point, you might notice that the VPN server works very much like a NAT box. Recall how general forwarding can be in an OpenFlow router. One can match on all kinds of packet header fields and take a variety of actions. With the proper matching rules and actions, an OpenFlow router can function like a router, a link-layer switch, a firewall, and a NAT box. Now try if you can implement a VPN server using an OpenFlow router as well.

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	*	*	*	*	*

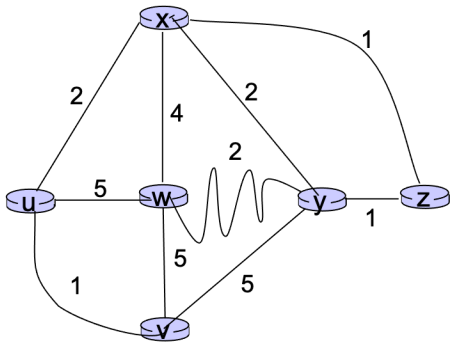
(1) Tell the matching rules for the HTTPS request packet. (2pt)
(2) Tell the actions to take for the HTTPS request packet. (2pt)
(3) Tell the matching rules for the HTTPS reply packet. (2pt)
(4) Tell the actions to take for the HTTPS reply packet. (2pt)

Sample Solution:

(1) IP Src == 234.56.78.9, TCP sport == 3345
(2) IP Src = 123.45.67.8, TCP sport = 613
(3) IP Dst == 123.45.67.8, TCP dport == 613
(4) IP Dst = 234.56.78.9, TCP dport =3345

(Grading policy: -1pt each mis-specifying src/dst or IP add/port#)

7. (ch52, 10pt) Run Dijkstra algorithm on the following graph.



Start from including node u to N'. At step 0, N'={u} and (D,p) pairs of node v, w, x, y, and z = {(1,u), (5,u), (2,u), inf, inf}. Iterate through all the nodes and complete the table below.

		(D,p)				
Step	N'	v	w	x	y	z
0	u	(1,u)	(5,u)	(2,u)	∞	∞
1						
2						
3						
4						
5						

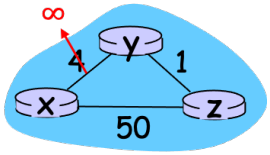
(1) Tell N' and (D,p) pairs of node v, w, x, y, z of step 1. (3pt)
(2) Tell N' and (D,p) pairs of node v, w, x, y, z of step 2. (3pt)
(3) Tell N' and (D,p) pairs of node v, w, x, y, z of step 3. (2pt)
(4) Tell N' and (D,p) pairs of node v, w, x, y, z of step 4. (2pt)

Sample Solution:

(1) uv, {(-,-), (5,u), (2,u), (6,v), inf}
(2) uvx, {(-,-), (5,u), (-,-), (4,x), (3,x)}
(3) uvxz, {(-,-), (5,u), (-,-), (4,x), (-,-)}
(4) uvxzy, {(-,-), (5,u), (-,-), (-,-), (-,-)}

(Grading policy: -1pt each for any missing or false N'/(D,p) entry)

8. (ch52, 14pt) Bad news travel slow in distance vector routing. Let's find out how many iterations it takes to converge when the link cost of (x,y) increases from 4 to infinity in the following network.



The Dx, Dy, and Dz before the link cost change are shown below. Now continue on computing the distance vectors after the link cost change and answer the following questions.

node x table

from \ cost to	x	y	z
x	0	~~4~~	~~5~~
y	4	0	1
z	5	1	0

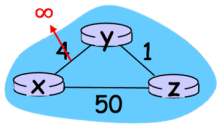
t0

from \ cost to	x	y	z
x	0	?	?
y	4	0	1
z	5	1	0

node y table

from \ cost to	x	y	z
x	0	4	5
y	~~4~~	0	~~1~~
z	5	1	0

	x	y	z
x	0	4	5
y	?	0	?
z	5	1	0



node z table

from \ cost to	x	y	z
x	0	4	5
y	4	0	1
z	5	1	0

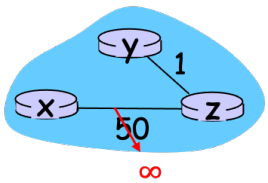
(1) Tell Dx at t0. (2pt)
(2) Tell Dy at t0. (2pt)
(3) Tell Dz at t1. (2pt)
(4) Tell Dy at t2. (2pt)
(5) Tell Dz at t43. (1pt)
(6) Tell Dy at t44. (1pt)
(7) Tell Dz at t45. (1pt)
(8) Tell Dy at t46. (1pt)
(9) Tell Dz at t47. (1pt)
(10) Tell Dy at t48. (1pt)

Sample Solution:

(1) Dx=(0,**51**,**50**)
(2) Dy=(**6**,0,**1**)
(3) Dz at t1 = (**7**,**1**,0)
(4) Dy at t2 = (**8**,0,**1**)
(5) Dz at t43 = (**49**,1,0)
(6) Dy at t44 = (**50**,0,1)
(7) Dz at t45 = (**50**,1,0)
(8) Dy at t46 = (**51**,0,1)
(9) Dz at t47 = (**50**,1,0)
(10) Dy at t48 = (**51**,0,1)

(Grading policy: each of the numbers in bold face counts 1pt)

9. (ch52, 15pt) Bad news travel slow is also referred to as the count to infinity problem. Let's find out if the distance vectors may take forever to converge considering the following network where the link cost of (x,z) increases from 50 to infinity.



The Dx, Dy, and Dz before the link cost change are shown below. Now continue on computing the distance vectors after the link cost change and answer the following questions.

node x table

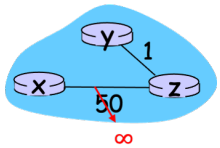
from \ cost to	x	y	z
x	0	~~51~~	~~50~~
y	51	0	1
z	50	1	0

t0

from \ cost to	x	y	z
x	0	?	?
y	51	0	1
z	50	1	0

node y table

from \ cost to	x	y	z
x	0	51	50
y	51	0	1
z	50	1	0



node z table

from \ cost to	x	y	z
x	0	51	50
y	51	0	1
z	~~50~~	~~1~~	0

	x	y	z
x	0	51	50
y	51	0	1
z	?	?	0

(1) Tell Dx at t0. (2pt)
(2) Tell Dz at t0. (2pt)
(3) Tell Dy at t1. (2pt)
(4) Tell Dz at t2. (2pt)
(5) Tell Dy at t3. (2pt)
(6) Tell Dz at t4. (2pt)
(7) Are Dy and Dz converging? (1pt)
(8) Why or why not? (2pt)

Sample Solution:

(1) Dx at t0 = (0, **inf**, **inf**)
(2) Dz at t0 = (**52**, **1**, 0)

(3) Dy at t1 = (**53**, 0, **1**)
(4) Dz at t2 = (**54**, **1**, 0)
(5) Dy at t3 = (**55**, 0, **1**)
(6) Dz at t4 = (**56**, **1**, 0)
(7) no, the value of $d_y x$ and $d_z x$ will count to infinity over time.
(8) whatever makes sense
E.g., c(x, z) used to be 50. y and z will be under the impression that there's a path to x at a finite cost (51 for y to x). As link x-z breaks, z will take whatever the best route informed by the neighbors previously. z will therefore choose to hop via y for a (finite cost) path to reach x. y will do the same and hop via z to get to x. z and y will never realize their best path to x are in fact infinite cost and will never converge to a stable DV table each.

(Grading policy for (1)-(6): each of the numbers in bold face counts 1pt)

10. (PA, 13pt) Go on to the PA server and log in with the exam username and password. Create a subdirectory `exam3` If you have not yet done so. Go to the `exam3` subdirectory and work on the following.

    Let's see if you've picked up the basics of programing a secure socket by implementing a very simple secure file upload service. To begin with, create secure-client.go (as below) in your `exam3` subdirectory. To execute the secure-client.go, you will need to have the client.cer and client.key in the same directory. They are copyable by all on the PA server here: /Users/client.cer and /Users/client.key.

    secure-client.go

```
package main

import "fmt"
import "bufio"
import "crypto/tls"

func check(e error) {
    if e != nil {
        panic(e)
    }
}

func main() {
    cert, err := tls.LoadX509KeyPair("client.cer", "client.key")
    check(err)
    //skip checking the certificate
    config := tls.Config{Certificates: []tls.Certificate{cert}, \
    InsecureSkipVerify: true}
    conn, _ := tls.Dial("tcp", "127.0.0.1:<your port#>", &config)
    defer conn.Close()

    writer := bufio.NewWriter(conn)
    len, errw := writer.WriteString("Hello World!\n")
    check(errw)
    fmt.Printf("Send a string of %d bytes\n", len)
    writer.Flush()
```

```
    reader := bufio.NewReader(conn)
    message, errr := reader.ReadString('\n')
    check(errr)
    fmt.Printf("Server replies: %s", message)
}
```

(1) Create e3-p10-1.go such that the server can receive the greeting from secure-client.go, send the number of bytes received back to secure-client.go, and then terminate. (5%)

(2) Create e3-p10-2.go such that the server can receive a series of messages from a client until a special line – ".\n". Then, return the total number of bytes received before the special ending line. (4%)

(3) Create e3-p10-3.go based on secure-client.go such that the client prompts for a filename, reads the file, and uploads a file to your e3-p10-2.go server. (4%)