Networks and Protocols
International University Bremen
Dr. Jürgen Schönwälder

Course: 320301
Date: 2005-12-21
Deadline: 2005-12-21

Final Examination

Name:	

Problem	F.1	F.2	F.3	F.4	F.5	F.6	Total
Maximum	10	20	20	20	20	10	100
Reached							
Grader							

Guidelines

- Write down your name — no name means no grade.
- Remember the code of academic integrity.
- Work first on the problems which you find easy to do.
- Keep in mind that not all problems have the same weight.
- Make sure you deliver all sheets at the end of the exam.

Good luck and have a nice and relaxing Christmas break!

/js

Problem F.1: *routing protocols* (2+2+2+2+2=10 points)

Indicate which of the following statements are correct or incorrect by marking the appropriate boxes. For every correctly marked box, you will earn two points. For every incorrectly marked box, you will loose one point. Statements which are not marked or which are marked as true and false will be ignored. The minimum number of points you can achieve is zero.

true	false	
☐	☐	Link-state routing protocols can suffer from the count-to-infinity problem.
☐	☐	The count-to-infinity problem can be solved by introducing split-horizon, poisoned-reverse, and strict synchronization of the routing updates.
☐	☐	Link-state routing protocols can have slow convergence time since the propagation of link state advertisements may lead to temporary routing inconsistencies.
☐	☐	Every transit Autonomous System (AS) is also a multi-homed AS.
☐	☐	The Border Gateway Protocol (BGP) does not suffer from the count-to-infinity problem because it keeps track of the full AS paths rather than just maintaining the cost to every destination.

Solution:

true	false	
☐	☒	Link-state routing protocols can suffer from the count-to-infinity problem.
☒	☐	The count-to-infinity problem can be solved by introducing split-horizon, poisoned-reverse, and strict synchronization of the routing updates.
☒	☐	Link-state routing protocols can have slow convergence time since the propagation of link state advertisements may lead to temporary routing inconsistencies.
☒	☐	Every transit Autonomous System (AS) is also a multi-homed AS.
☒	☐	The Border Gateway Protocol (BGP) does not suffer from the count-to-infinity problem because it keeps track of the full AS paths rather than just maintaining the cost to every destination.

Problem F.2: *protocol messages* (20 points)

The following frame (shown in hexadecimal notation) was captured on an Ethernet yesterday.

```
00 40 63 d8 10 92 00 0d 93 c2 41 6c 86 dd 60 00
00 00 00 28 06 40 20 01 06 38 07 09 00 05 02 0d
93 ff fe c2 41 6c 20 01 06 38 07 09 00 04 00 00
00 00 00 00 00 04 c0 a1 00 16 14 8b b0 6d 00 00
00 00 50 02 ff ff 74 e5 00 00
```

Decode the packet. Fill in a table which has the following format:

offset	bytes	meaning
0	00 . . .	. . .

The offset column identifies the byte in which a field starts. The first byte has offset 0. What is the purpose of this message?

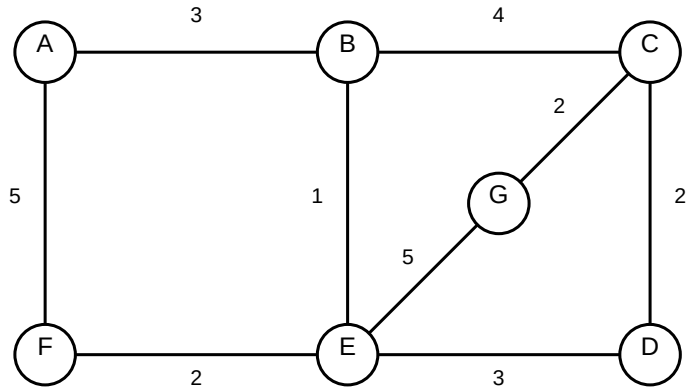
Solution:

offset	bytes	meaning
0	`0040 63d8 1092`	Ethernet destination address
6	`000d 93c2 416c`	Ethernet source address
12	`86dd`	Ethernet type (0x86dd = IPv6)
14	`6`	IP version 6
14	` 00`	IP traffic class 0
15	`   0 0000`	IP flow label 0
18	`0028`	IP packet length 0x28 = 40 byte
20	`06`	IP Next header (0x06 = TCP)
21	`40`	IP hop limit 0x40 = 64
22	`2001 0638 0709 0005 020d 93ff fec2 416c`	IP source address
38	`2001 0638 0709 0004 0000 0000 0000 0004`	IP destination address
54	`c0a1`	TCP source port 0xc0a1 = 49313
56	`0016`	TCP destination port 0x0016 = 22
58	`148b b06d`	TCP sequence number
62	`0000 0000`	TCP acknowledgment number
66	`5`	TCP header 0x5 * 4 = 20 bytes
66	` 0`	TCP reserved
67	`  02`	TCP flags 0x02 = SYN
68	`ffff`	TCP window size
70	`74e5`	TCP checksum
72	`0000`	TCP urgent pointer

The packet is the beginning of a handshake to establish a TCP over IPv6 connection to an SSH server running on `2001:638:709:4::4`.

Problem F.3: *link state routing* (10+10=20 points)

Consider the following network topology:



a) Determine the shortest-paths from node G to all other nodes using Dijkstra's algorithm. For each step, fill a row in a table structured as follows:

Step	A	B	C	D	E	F	G	Current	Permanent
0	∞	∞	∞	∞	∞	∞	0		
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									
12									
13									
14									
15									
...									

b) Link-state routing protocols use flooding as a means to distribute link state advertisements. Each link-state advertisement contains a sequence number and routers keep track of all the (source router, sequence number) pairs that they see. When a link state packet comes it, it is checked against the list of packets already seen. If it is new, it is forwarded on all lines except the one it arrived on. If it is a duplicate, it is discarded. If a packet with a lower sequence number is seen, it is reject as being obsolete since the router has more recent data.

This basic algorithm has some minor problems:

1. Sequence numbers could wrap around.
2. A crashing router might loose its sequence number and after reboot start with 0 again.
3. A corrupted sequence number (e.g., the most significant bit swaps from 0 to 1) can render large portions of the sequence number space unusable.

Design a solution which addresses these problems. If you have ideas for multiple solutions, discuss their strengths and weaknesses.

Solution:

a) Below is the table produced by executing Dijkstra's algorithm:

Step	A	B	C	D	E	F	G	Current	Permanent
0	∞	∞	∞	∞	∞	∞	0		
1	∞	∞	2(G)	∞	5(G)	∞	0	G	G
2	∞	6(C)	2(G)	4(C)	5(G)	∞	0	C	C,G
3	∞	6(C)	2(G)	4(C)	5(G)	∞	0	D	C,D,G
4	∞	6(C)	2(G)	4(C)	5(G)	7(E)	0	E	C,D,E,G
5	9(B)	6(C)	2(G)	4(C)	5(G)	7(E)	0	B	B,C,D,E,G
6	9(B)	6(C)	2(G)	4(C)	5(G)	7(E)	0	F	B,C,D,E,F,G
7	9(B)	6(C)	2(G)	4(C)	5(G)	7(E)	0	A	A,B,C,D,E,F,G

b) The first problem can be solved by simply making the sequence number space large enough. The second can be solved by adding a boot counter to the sequence number. The third problem can be solved by using checksums to verify that the sequence number is correct.

An alternative solution is to use soft-state and to include a validity interval in a link state advertisement. Once the advertisement expires, the state is removed and processing can continue. The soft-state solution also deals with routers that disappear after failure or reconfiguration and is thus preferable, although the reaction time depends on how the validity is chosen.

Problem F.4: *transmission control protocol* (10+10=20 points)

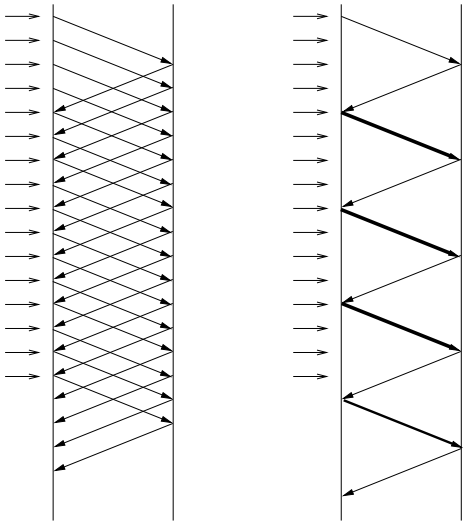
A secretary types characters into an TCP connection (each character is represented by a single byte). The secretary types with a constant speed of 1 character every 100 ms and 100 characters in total. Compare the performance of a TCP connection that uses the Nagle algorithm with the performance of a TCP connection where the Nagle algorithm is disabled.

a) With the aid of a time sequence diagram, explain how the Nagle algorithm works in this scenario.

b) What is the overall bandwidth consumed if the TCP connection runs over IPv4/IEEE802.3? What happens if we use IPv6 instead of IPv4? (Assume that no options are used and ignore any TCP connection setup and tear-down overhead.)

Solution:

a) The Nagle algorithm handles situations where data arrives one byte at a time. The first byte is send immediately while all subsequent bytes are buffered until the byte in flight has been acknowledged.



This algorithm provides noticeable improvements especially for interactive traffic where a quickly typing user is connected over a rather slow network. It can also saves quite some bandwidth as it reduces the number of messages that are being exchanged.

b) Let f be the size of the link-layer header (14 bytes in case of Ethernet, excluding synchronization and checksum bytes). Similarly, let n be the size of the network layer header (20 bytes in case of IPv4 and 40 bytes in case of IPv6) and t be size of the transport layer header (20 bytes in case of TCP without options).

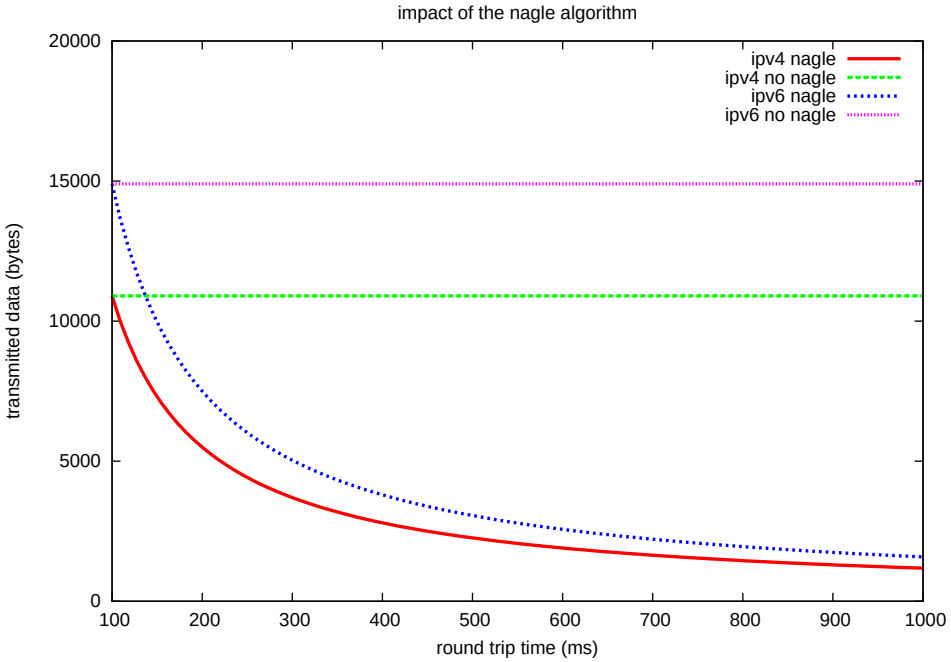
Furthermore, let d be the delay in which data arrives and let r be the round trip time. The number of data bytes to transmit is given by N.

Without Nagle, every arriving byte is send as a separate TCP segment which is then acknowledged by the receiver. This leads to $B = N \cdot (D + A)$ with $D = f + n + t + 1$ and $A = f + n + t$ where D is the size of a data segment and A is the size of an acknowledgment segment.

With the Nagle algorithm enabled, we will get the same behaviour as long as $d \leq r$. However, if $d > r$, then a number of data bytes will be buffered. Thus, every data segment now contains $D = f + n + t + \lfloor \frac{r}{d} \rfloor$ bytes. and we need only $N / \lfloor \frac{r}{d} \rfloor$ data packets. (Note that this equation is a simplification since it does not handle the first and last exchange correctly.)

With $N = 100$, $d = 100\,ms$, $f = 14\,byte$, $t = 20\,byte$ and $r = 400\,ms$, we get the following numeric values:

	IPv4 ($n = 20$)	IPv6 ($n = 40$)
no Nagle (bytes)	10900	14900
Nagle (bytes)	2800	3800


impact of the nagle algorithm
transmitted data (bytes)
round trip time (ms)
ipv4 nagle
ipv4 no nagle
ipv6 nagle
ipv6 no nagle
0
5000
10000
15000
20000
100
200
300
400
500
600
700
800
900
1000

Problem F.5: *data encoding* (10+10=20 points)

a) A binary file is 3070 bytes long. How long will it be if encoded using base64 encoding, with a CR+LF pair inserted after every 76 bytes and at the end of the file?

b) The XDR standard allows applications to exchange data in a portable way, independent of the local data representation. Consider the following XDR data definition:

```
const MAXUSERNAME = 32;     /*  max length of a user name  */
const MAXFILELEN = 65535;   /*  max length of a file       */
const MAXNAMELEN = 255;     /*  max length of a file name  */

enum filetyp {
    TEXT = 0,       /*  ascii data  */
    DATA = 1,       /*  raw data    */
    EXEC = 2        /*  executable  */
};

struct file {
    string filename<MAXNAMELEN>; /*  name of file  */
    filetype type;               /*  info about file  */
    string owner<MAXUSERNAME>;   /*  owner of file    */
    opaque data<MAXFILELEN>;     /*  file data        */
};
```

User `john` stores a simple LISP program which just contains the LISP expression `(quit)` in a file named `useless`. Show the encoding of this file according to the XDR definition shown above.

Solution:

a) Base64 encodes 3 binary input bytes in 4 output bytes. A line of 76 output bytes can therefore hold $76 \cdot 3/4 = 57$ binary input bytes. Hence, a total of

$$L = \left\lceil \frac{3070}{57} \right\rceil = 54 \text{ lines}$$

lines is needed, where the last line represents only $3070 - 53 \cdot 57 = 49$ binary input bytes. Since 49 is not a multiple of 3, two additional fill bytes must be added and two `=` characters must be appended to indicate that two fill bytes were used.

Summing up, we have 53 lines with $76 + 2$ bytes and 1 line with $51\frac{3}{4} = 68$ bytes plus 2 `=` characters and every line has two line end characters. This leads to:

$$B = 53 \cdot (76 + 2) + 1 \cdot (68 + 2 + 2) = 4206 \text{ bytes}$$

b) Since XDR is 32-bit aligned, we list the contents in terms of 32-bit words.

offset	contents	description
0	0x00000007	length of the file name (7 bytes)
4	0x7573656c	file name “usel”
8	0x65737300	file name “ess” + padding
12	0x00000002	file type EXEC
16	0x00000004	length of the owner name (4 bytes)
20	0x6a6f686e	owner name “john”
24	0x00000006	file content length (6 bytes)
28	0x28717569	file content “(qui”
32	0x74290000	file content “t)” + padding

Problem F.6: *hypertext transfer protocol* (2+2+2+2+2=10 points)

Indicate which of the following statements are correct or incorrect by marking the appropriate boxes. For every correctly marked box, you will earn two points. For every incorrectly marked box, you will loose one point. Statements which are not marked or which are marked as true and false will be ignored. The minimum number of points you can achieve is zero.

true	false	
☐	☐	With a non-persistent connections between a browser and an origin server, it is possible for a single TCP segment to carry two distinct HTTP request messages.
☐	☐	The `Date:` header in the HTTP response indicates when the object in the response was last modified.
☐	☐	Two distinct web pages (for example `http://www.iu-bremen.de/research/` and `http://www.iu-bremen.de/academics/`) can be sent over the same persistent connection.
☐	☐	When a user requests a web page which consists of some text and two images, the browser will send one request message and receive three response messages.
☐	☐	Both, the client and the server, can signal the closing of a persistent connection.

Solution:

true	false	
☐	☒	With a non-persistent connections between a browser and an origin server, it is possible for a single TCP segment to carry two distinct HTTP request messages.
☐	☒	The `Date:` header in the HTTP response indicates when the object in the response was last modified.
☒	☐	Two distinct web pages (for example `http://www.iu-bremen.de/research/` and `http://www.iu-bremen.de/academics/`) can be sent over the same persistent connection.
☐	☒	When a user requests a web page which consists of some text and two images, the browser will send one request message and receive three response messages.
☒	☐	Both, the client and the server, can signal the closing of a persistent connection.

Appendix

The hexadecimal ASCII set:

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 ‘	23 #	24 $	25 %	26 &	27 ’
28 (	29)	2a *	2b +	2c ,	2d -	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [	5c \	5d]	5e ^	5f _
60 ‘	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c |	7d }	7e ~	7f del

TCP header: