Chalmers University of Technology
Computer Science and Engineering
Elad Michael Schiller

2014-10-31

Written exam in EDA387/DIT663 Computer Networks 2014-10-31. Exam time: 4 hours.

Means allowed: Nothing except paper, pencil, pen and English - xx dictionary.

Examiner: Elad Michael Schiller, phone: 073-6439754
Note that student questions can be answered only by phone.

Credits:	30-38	39-47	48-Max
Grade:	3	4	5
Grade (GU)	G	G	VG

1. The answer must be written in English (even for Swedish students). Use proper grammar and punctuation.
2. All answers need to be motivated, unless otherwise stated. Correct answers without motivation or with wrong motivation will not be given full credit.
3. Answer concisely, but explain all reasoning. Draw figures and diagrams when appropriate.
4. Write clearly. Unreadable or hard-to-read handwriting will not be given any credit.
5. Do not use red ink.
6. Solve only one problem per page.
7. Sort and number pages by ascending problem order.
8. Anything written on the back of the pages will be ignored.
9. Do not hand in empty pages or multiple solutions to the same problem. Clearly cross out anything written that is not part of the solution.

Question 1 DNS (8 points)

Please answer each of the sub-questions given below separately and by using **DNS-terminology and concepts.**

1a. (4p) Mention and describe the **meaning and contents** of at least **four** commonly used Resource Records (name, type, value) in the DNS database.

1b. (4p) Suppose that you are using the Chalmers network to connect your laptop to the Internet. Suppose also that you want to access the web site www.tue.nl for the **first** time. Explain **how and why** DNS will be involved immediately after entering the name of the site in your browser. Assume that there is **no** cached DNS-information (about this site), anywhere in Chalmers network. The answer should, specifically and technically, explain the necessary operation, including:

- the interaction and communication between the different DNS resolvers and servers,
- the protocols and messages used, and
- the final outcome.

Question 2 IPv6 Addresses (6 points)

These three addresses are given with IPv6 representation:

(i) `2001:6b0:2:10::1`
(ii) `FF02::1:ff6c:14dd`
(iii) `FE80::20c:f1ff:fe6c:14dd`

Please answer the following sub-questions in relation to the above addresses.

2a. (1p) Decompress and rewrite each of the given addresses showing all hexadecimal digits.

2b. (2p) What is the "type" of each of these IPv6 addresses? Explain what each type does imply.

2c. (1p) Which of the given addresses cannot be used as valid source address in IPv6 packet? Explain why?

2d. (2p) What is the "scope" of each of these IPv6 addresses? Explain what each scope does imply.

Question 3 ICMPv6 (8 points)

3a. (2p) What is the **main** purpose of IPv6 Neighbor Discovery? Explain **clearly** the operation.

3b. (2p) What are the messages deployed in IPv6 Neighbor Discovery? Explain how these messages will be encapsulated and addressed in layer-2 and layer-3 PDUs (i.e. packets and frames).

3c. (4p) What is the purpose of sending the message "Router Advertisement"? What are the **most** important parts of information does it contain? Explain at least **three** and how they are useful for IPv6 nodes.

Question 4 (6 points) Socket API: `select()`

Each of the following parts of a program contains a flaw. Identify and describe the flaw in a few short sentences or points. You do not have to correct the flaw; you should just find and describe it! (Note: you're not looking for, e.g., syntax errors. Find conceptual flaws in the program.)
Hint: The program uses `select()` and they are supposed to be non-blocking. Consider which operations can actually block the processes that execute these programs.
The following program accepts new connections using the `listenfd` socket. The first byte sent by a client is expected to be an 8 bit ID.

- You may assume that the `handle_*_error()` methods do something sensible.
- The helper method `register_client(client, id)` verifies the client ID is acceptable and if that is the case, enters the client into a global list. Otherwise it closes the connection.
- The method `add_client_sockets_to_readfds()` properly adds all active clients in the global list to the readfds. It returns the largest socket number it encounters.
- `handle_registered_clients()` handles clients that are ready to send data according to `readfds`, and removes clients that close their associated connections from the global list. No data is ever sent to the clients, the program only receives and processes data sent to it.

```
/* includes, declarations, etc. */
int main() {
	int listenfd = -1;
	/* initialization code, such as setting up a listening socket on listenfd, has been omitted - this is not the error you're looking for */
	while( 1 ) {
		fd_set readfds; // initialize read set
		FD_ZERO( &readfds );
		int maxfd = add_client_sockets_to_readfds( &readfds );
		FD_SET( listenfd, &readfds );
		if( listenfd > maxfd ) maxfd = listenfd;
		int ret = select( maxfd+1, &readfds, 0, 0, 0 ); // call select
		if( -1 == ret ) handle_select_error();
		// is there a new client waiting?
		if( FD_ISSET( listenfd, &readfds ) ) {
			sockaddr_in clientAddr;
			socklen_t clientAddrLen = sizeof(clientAddr);
			int client = accept( listenfd,
				(sockaddr*)&clientAddr,
				&clientAddrLen
			);
			if( -1 == client ) handle_accept_error();
			// receive 8bit client ID
			unsigned char id;
			int ret = recv( client, &id, sizeof(id), 0 );
			if( 0 == ret ) {
				close( client );
				continue;
			}
			if( -1 == ret ) handle_recv_error();
			// register client
			register_client( client, id );
		}
		handle_registered_clients(&readfds);//handle registered clients
	}
	return 0;
}
```

Question 5 (8 points)

We learned in class a self-stabilizing algorithm for BFS spanning tree construction, see the code below. Explain how transient faults can cause the system to output an error. We define a *floating distance* in configuration c, as a value stored in $r_{ij}.dis$ that is smaller than the distance of p_i from the root, where *dis* is the distance field of the registers.

```
01 Root: do forever
02         for m := 1 to δ do write r_im := (0,0)
03     od
04 Other: do forever
05         for m := 1 to δ do lr_m := read(r_mi)
06         FirstFound := false
07         dist := 1 + min{lr_m.dis | 1 ≤ m ≤ δ }
08         for m := 1 to δ
09         do
10             if not FirstFound and lr_m.dis = dist -1
11                 write r_im := (1,dist)
12                 FirstFound := true
13             else
14                 write r_im := (0,dist)
15         od
16     od
```

Prove that for every $k > 0$ and for every configuration that follows $\Delta + 4k\Delta$ rounds, it holds that:

- If there exists a floating distance, then the value of the smallest floating distance is at least k.
- The value in the registers of every processor that is within distance k from the root is equal to its distance from the root.

Proof. Note that in every 2Δ successive rounds, each processor reads the registers of all its neighbors and writes to each of its registers. We prove the lemma by (1) induction for k.

Base Case: Proof for k=1. Distances stored in the registers and internal variables are non-negative; thus the value of the smallest floating distance is at least 0 in the first configuration. During the first 2Δ rounds, each non-root processor p_i, computes the value of the variable *dist* (line 7). The result of each such computation must be (2) [illegible]. Let c_2 be the configuration reached following the first computation of the value of *dist* by each processor.

Each non-root processor writes to each of its registers the computed value of *dist* during the 2Δ rounds that follow c_2. Thus, in every configuration that follows the first 4Δ rounds there is no non-root processor with value 0 in its registers. The above proves (3) Base Case (k=1)

To prove (4) Root processor, note that the root repeatedly writes the (5) <0,0> to its registers in every (6) 4Δ rounds. Let c_1 be the configuration reached after these (7) 4Δ rounds. Each processor reads the registers of the root and *then* writes to its own registers during the 4Δ rounds that follow (8) first round. In this write operation the processor assigns (9) 1+1 to its own registers. Any further read of the root registers returns the value (10) <1, dist> therefore, the value of the registers of each neighbor of the root is (11) <0,1> following the first $\Delta + 4\Delta$ rounds. Thus, (11) 2 holds as well.

Induction Step. We assume correctness for k (12) = 0 and prove for k + 1. Let $m \geq k$ be the smallest floating distance in the configuration c_{4k} that follows the first $\Delta + 4k\Delta$ rounds. During the 4Δ rounds that follow c_{4k}, each processor that reads m and chooses m as the smallest value assigns (13) k to its distance and writes this value. Therefore, the smallest floating distance value is $m + 1$ in the configuration $c_{4(k+1)}$. This proves (14) k+1.

Since the smallest floating distance is m (15) ≥ k, it is clear that each processor reads the distance of a neighboring processor of distance k and assigns (16) k+1 to its distance. ■

Question 6 (4 points)

6.a **(1 p)** The set of legal executions, *LE*, includes all executions in which the system behaves according to the required properties (and no other execution). Use the notation *LE* to define the term *safe configuration*. We say that configuration *c* is *safe* if every ~~state~~ execution is LE.

6.b **(3 p)** We learned in class a non-stabilizing algorithm for synchronous consensus, see the code below. Explain how transient faults can cause the system to output an error.

```
initialization
    pulse_i := 0
    O_i := I_i
while pulse_i ≤ D do
    upon a pulse
        pulse_i := pulse_i + 1
        send (O_i)
        forall P_j ∈ N(i) do receive (O_j)
        if O_i = 0 and ∃ P_j ∈ N(i) | O_j = 1 then
            O_i := 1
```

This algorithm is not a self stabilizing algorithm

Question 7 (6 points)

Please find below a self-stabilizing algorithm for leader election, where *N* is an upper bound on the number of processors in the system.

7.a **(2 p)** Please define the safe configuration of the algorithm. Make sure that you consider all variables and shared registers.

7.b **(4 p)**. Suppose the system execution, *R*, starts in a safe configuration, *c*. Let a_i be a step that processor *pi* takes immediately after *c* and just before c'. Please show that c' is safe.

```
do forever
    (candidate,distance) = ( ID(i), 0 )
    forall P_j ∈ N(i) do
      begin
        (leader_i[j],dis_i[j]) := read( leader_j, dis_j )
        if (dis_i[j] < N) and ((leader_i[j] < candidate) or
           ((leader_i[j] = candidate) and (dis_i[j] < distance))) then
              (candidate,distance) := ( leader_i[j],dis_i[j] + 1)
      end
    write (leader_i ,dis_i) := (candidate,distance)
od
```

Question 8 (8 points)

We learned in class several algorithms for self-stabilizing clock synchronization. Please find below the code of a couple of them, which we call: converge-to-the-min and -max.

Converge-to-the-max

```
upon a pulse
    forall P_j ∈ N(i) do send (j, clock_i)
    max := clock_i
    forall P_j ∈ N(i) do
        receive(clock_j)
        if clock_j > max then max := clock_j
    od
    clock_i := (max + 1) mod ((n+1)d + 1)
```

Converge-to-the-min

```
upon a pulse
    forall P_j ∈ N(i) do send (j, clock_i)
    min := clock_i
    forall P_j ∈ N(i) do
        receive(clock_j)
        if clock_j < min then min := clock_j
    od
    clock_i := (min + 1) mod (2d − 1)
```

8.a (2 p) What do the constants d and n stand for?
8.b (1 p) Please compare these two algorithms with respect to their scalability property. Which one scales better? Why?
8.c (1 p) Please compare these two algorithms with respect to the service provided to the application layer. Which one is easier to work with? Why?
8.d (4 p) Please complete the correctness proof of the algorithm converge-to-the-min

Suppose that no processor (1) [frasit] during the first (2) [2d] pulses. Then we can use simple (3) [induction] to show that synchronization is achieved. Otherwise, a processor (4) ____ during the first (5) ____ pulses. Therefore, (6) ____ pulses after this point a configuration c is reached, such that there is no clock value greater than (7) [2d+1]: the first (8) [round( ].

clock = (min+

Question 9 (6 points)

9.a (2 p) Define the task of wait-free self-stabilizing clock synchronization. Given a fixed integer *k*, once a processor p_i works correctly for at least *k* time units and continues working correctly, the following properties hold:

- Adjustment: p_i does not (1)____ its clock.
- Agreement: p_i's clock (2)____ with the clock of (3) ____ that has also (4) ____ for at least *k* time units.

9.b (4 p) We learned in class an algorithm for wait-free self-stabilizing clock synchronization for the fully connected graph, please find below its code. Each processor P has the following two variables: (1) P.clock $\in \{0 \ldots M\text{-}1\}$ and (2) $\forall$ Q : P.count[Q] $\in \{0,1,2\}$. We say that processor P is behind Q if P.count[Q]+1 (mod 3) = Q.count[P].

Suppose the processor P executes more than k=2 successive steps. Show that the set *NB*, which is *R* in the code to the right, is not empty following P's first step.

The program for P:

1) Read every count and clock
2) Find the processor set R that are not behind <u>any</u> other processor
3) If R $\neq \varnothing$ then P finds a processor K with the maximal clock value in R and assigns P.clock := K.clock + 1 (mod M)
4) For every processor Q, if Q is not behind P then P.count[Q] := P.count[Q] + 1 (mod 3)

Question 10 (8 points)

10.a (2 p) Define the task of vertex coloring.

10.b (2 p) Please find below one of the self-stabilizing algorithms for vertex coloring that we learned in class. How long does it takes for the algorithm to convergence. Please give an example for a particularly long convergence period.

10.c (4 p) Does this algorithm guarantee the shortest convergence possible? In case you think that it is, then please give a formal proof for a matching lower bound. In case you think that it is not, please explain how to change the algorithm below so that the convergence time become shorter. (Say which variables needs to be added, rewrite the code and give an example in which the algorithm below takes a long time to converge and the one you write takes a very short time to converge.)

```
Do forever
      GColors := Ø
      For m:=1 to δ do
            lr_m:=read(r_m)
            If ID(m)>i then
                  GColors := GColors ∪ {lr_m.color}
      od
      If color_i ∈ GColors then
        color_i:=choose(\\ GColors)
      Write r_i.color := color_i
od
```