Chalmers University of Technology

Computer Science and Engineering
Elad Michael Schiller

2019/10/31

Written exam in EDA387/DIT663 Computer Networks 2019-10-31. Exam time: 4 hours.

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

Examiner: Elad Michael Schiller, phone: 073-6439754

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: Selecting the right phrases from a list of phrases (4 points)

(4 points) Four phrases are missing in Fig 1. Please complete the missing phrases using only four phrases from the following list of nine phrases. (a) timeouts, (b) slow start period, (c) fast retransmission, (d) two duplicate ACKs, (e) maximum segment size, (f) additive increase, (g) selective acknowledgment, (h) explicit congestion notification, and (i) probing for usable bandwidth. Choose only the most appropriate phrases.

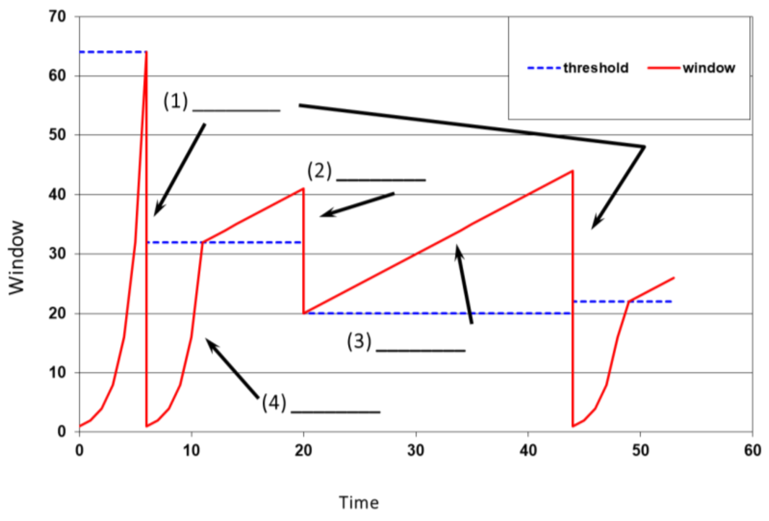
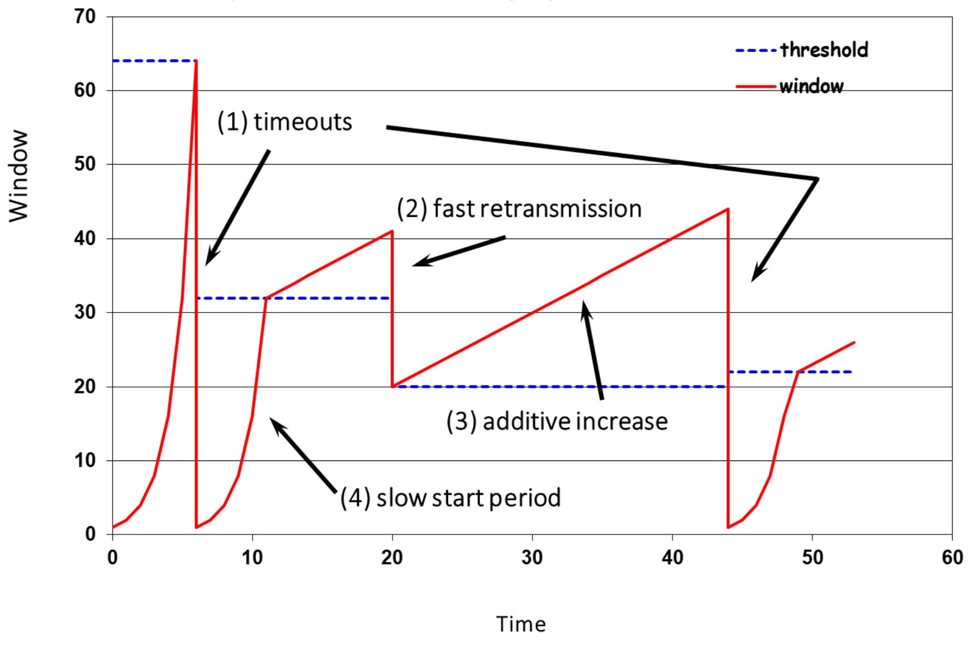


Figure 1 select the correct phrases

The solution is depicted in the following figure.



....

Question 2: true or false questions (18 points)

For each of the following statements, please state whether they are true or false. In case you are not sure about your answer or the statement is not clear to you, you are welcome to justify your claim briefly (but you don't have to).

2a. (3 points) The advantage of Additive Increase and Multiplicative Decrease (AIMD) is that: (i) when additively increasing, the ratios do not change, and (ii) when multiplicatively decreasing, the ratios indeed change and there is a fast response. Note that Fig. 2 might assist you.
Flase

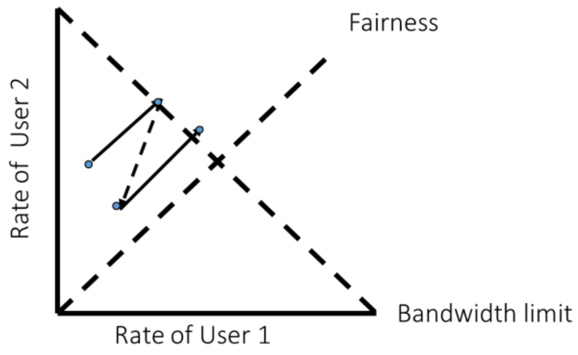


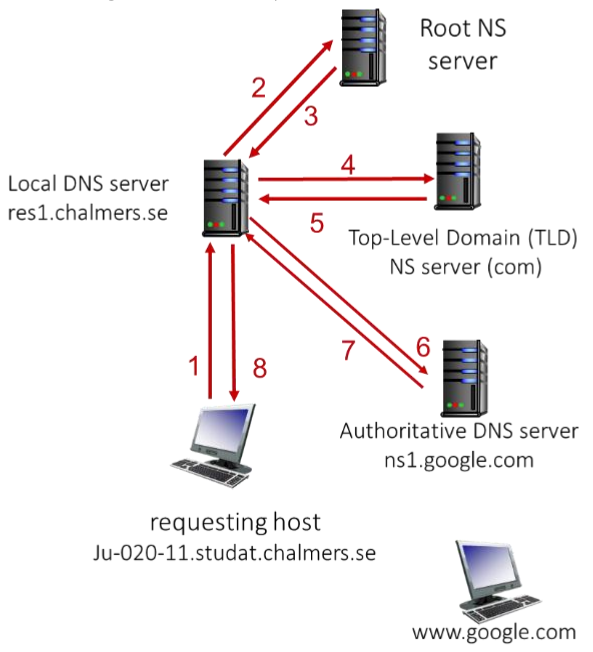
Figure 2 AIMD with two users over one link

2b. (3 points) The next sentences consider the key differences between flow control and congestion control in TCP/IP as studied in class. Flow control basically states that a network end-point can transmit only a certain number of packets and cannot add more packets to a network until an acknowledgment is received. Congestion control works by refusing new TCP/IP connections until congestion is resolved.
Flase

2c. (3 points) In the context of TCP/IP and socket API that were studied in class, a call to accept() can block if there are no client connections that became established, *i.e.*, completed the three-way handshake. Moreover, a call to read() can block if the receiver's buffer contains no messages to be read. Furthermore, a call to write() can block if there is no sufficient space left at sender's buffer. Note that a call to socket() cannot block, but it can return an error, for example, when the calling process has already too many open file descriptors.
True

2d. (3 points) A good design for the domain name system (DNS) for the Internet needs to allow frequent writes to the database as much as reads to it. Moreover, a good DNS design needs to provide strict consistency guarantees.
Flase

2e. (3 points) The domain name system (DNS) of the Internet is hierarchical. That is, it includes root servers at the top-level, then domain servers at the next level, and then subdomain server on the level that follows.
True. In case the answer said that the names are not exact, then this answer got between 2.5 and 3 points --- depending on the wordings of the justification.


Root NS server
2
3
4
5
Local DNS server res1.chalmers.se
Top-Level Domain (TLD) NS server (com)
1
8
7
6
Authoritative DNS server ns1.google.com
requesting host Ju-020-11.studat.chalmers.se
www.google.com
Figure 3 DNS queries and replies

2f. (3 points) *Glue records* help to significantly reduce the number of server-to-server interactions during a DNS query. Fig. 3 illustrates an example in which a host in "chalmers.se" wants the IP address for www.google.com. Upon the first occurrence of such query, four servers need to participate in the name-resolution process. However, if that query occurs frequently, the involved servers can ``glue'' the resource

records to their local memory. By that, the query answer can already be ready, or partially ready, at these servers. Thus, the number of interactions will be reduced.
Flase

…….

Question 3: **Non-self-stabilizing digital clock synchronization (8 points)**

3a. (1 point) Please define the task of digital clock synchronization by filling up the following sentences.
[agreement]: identical clock values
[progress]: the clock values are incremented every pulse

Fig. 4 contains a non-self-stabilizing algorithm for digital clock synchronization.

```
upon a pulse
    forall $P_j \in N(i)$ do send $(j, clock_i)$
    let $lClocks := \{clock_i\}$
    forall $P_j \in N(i)$ do
        receive($clock_j$)
        $lClocks := lClocks \cup \{clock_j\}$
    od
    $clock_i := 1 + max(lClocks)$
```

Figure 4 Non-self-stabilizing digital clock synchronization

3b. (1 point) Please explain why this algorithm is not self-stabilizing.

Because it does not have bounded size variables, i.e., the clock shared variable. Since the actual implementation will use finite size variables, self-stabilizing algorithms must include the specification of the bound, MAXINT, on the variable size that is safe to use. Note that in case of a single transient fault, MAXINT can be reached at once. Thus, if MAXINT is too small, there is no guarantee that the clock will ever satisfy the task requirement of agreement.

3c. (6 points) Please demonstrate that within $O(d)$ pulses, a system that run the algorithm presented in Fig. 4 satisfies the task of digital clock synchronization, where d is the graph diameter. Your proof should include all the key steps of the complete proof. Please assume that transient faults cannot occur (not even before the system starts running). Moreover, please use the assumption that were used in class.

A very similar proof was given in class and it also appears in Chapter 6.1 of the textbook.

Question 4: **Super-self-stabilizing colouring in a directed ring with a distinguished processor (18 points)**

4a. (1 point) please define the task of vertex colouring (for general graphs).

Every node has a colour that is not the same colour as any of his neighbours.

```
$P_0$:          do forever
                    $x_0 := (x_{n-1}+2) (\text{mod } 3)$
$P_i (i \neq 0)$:  do forever
                    $x_i := (x_{i-1}+1) (\text{mod } 2)$
```

Figure 5 self-stabilizing optimal coloring on a directed ring with a distinguished node

Let us consider the self-stabilizing algorithm presented in Fig. 5 for colouring of a directed ring with a distinguished node, p_0.

4b. (6 points) Please show that the algorithm in Fig. 5 stabilizes within $O(n)$ asynchronous cycles. Your proof needs to demonstrate all the key steps of the complete proof.

4c. (1 point) please explain the difference between self-stabilizing systems and super-self-stabilizing systems. Please provide clear statements using the definitions studied in class.

Written in the slides and Chapter 7.1 in the textbook.

For items 4d and 4e of this question, let us enhance the model of the directed ring that we have considered so far in this question. Suppose that the ring is bi-directional, *i.e.*, p_i can read also $x_{(i+1) \bmod n}$ rather than just read $x_{(i-1) \bmod n}$ as well as read and write x_i. Moreover, non-distinguished nodes can be removed from the ring. However, only one node at a time can be removed. Whenever an event of node removal occurs with respect to p_i, both of its neighbours, i.e. $x_{(i-1) \bmod n}$ and $x_{(i+1) \bmod n}$, raise interrupts with the parameter 'removedNeighbor=i'. This is depicted by the algorithm template presented in Fig. 6.

```
01  P_0:            do forever
02                      x_0 :=
03  P_i(i ≠ 0):     do forever
04                      x_i :=
05  interrupt section (newNeighbor)
06  P_0:   if removedNeighbor=i-1 then
07                  x_i := ______
08         else     x_i := ______
09  P_i(i ≠ 0): if removedNeighbor=i-1 then
10                  x_i := ______
11         else     x_i := ______
```

Figure 6 template for a super-self-stabilizing algorithm for coloring on a directed ring with a distinguished node

4d. (4 points) please complete (and if needed modify) the algorithm template presented in Fig. 6 so that it becomes super-self-stabilizing. In your answer, please provide the complete code of your solution. Moreover, please state your assumptions clearly.

Chapter 7.1 presents a super-self-stabilizing algorithm for vertex colouring. The algorithm assumes globally unique processor identifiers. Therefore, this algorithm does not fit here. There are transformation algorithms from a system with a unique identifier to the one that is given. However, the transformation learnt in class is not super-self-stabilizing. So instead, we tailor the solution of Chapter 7.1 is a way that is dedicated to the given bi-directional ring, see the code below.

```
01  P_0:          do forever
02                    x_0 := 3
03  P_i(i ≠ 0):   do forever
04                    if x_i = x_{i-1} ∨ x_i = ⊥ then x_i := min ({0,1,2}\{x_{i-1},x_{i+1}})
06  interrupt section (newNeighbor)
07  P_i(i ≠ 0):       if removedNeighbor=i-1 then x_i := ⊥
```

We assume that $x_i{:}i \neq 0$ can only encode the colors in {0,1,2}.

4e. (6 points) please prove the correctness of the algorithm that you have provided for item 4d. Your proof should include all the key steps of the complete proof. Moreover, please state your assumptions clearly.

To the end of proving that the algorithm is self-stabilizing, we show that, in the absence of transient faults and topological changes, the system reaches a safe configuration after which a legal execution starts.

Claim 1 (convergence): within 2(k+1) asynchronous cycles, the processors $p_0,\ldots,p_k$ have each a colour that is unique within their individual neighbourhoods. Moreover, their colours do not change (in the absence of transient faults and topological changes).

Proof: This proof is by induction on p_k's distance, k, from the distinguished processor, p_0.

For the case of k=0, we note that x_0=3 within a single asynchronous cycle due to line 02, and that the variable color cannot change. Moreover, since only p_{n-1} cannot encode the colour 3 in x_{n-1}, it holds that p_0 has a colour that is unique to its neighbourhood.

Suppose that the claim is true for every k>0. We show that the claim is true for the case of k+1.

Let $c_{2(k+1)}$ be the configuration the system reaches within 2(k+1) asynchronous cycles. By the induction hypothesis, $p_0,\ldots,p_k$ have each a colour that is unique within their individual neighbourhoods and these colours do not change in $c_{2(k+1)}$ (due to the absence of transient faults and topological changes). We argue that, within two asynchronous cycle, p_{k+1} have a colour that is unique within its neighbourhood, p_k and $p_{k+2 \bmod n}$, and that colour does not change (in the absence of transient faults and topological changes).

We start by considering the case of k<n-2. Suppose that $x_k = x_{k+1}$ *in* $c_{2(k+1)}$. Within one asynchronous cycle, p_{k+1} select a colour that is different than the one of p_k (due to line 04). Once that happens, the if-statement conditions of line 04 cannot hold again for p_{k+1}. Suppose that $x_{k+1} = x_{k+2}$ *in* $c_{2(k+1)+1}$. Within one asynchronous cycle, p_{k+2} selects a colour that is different than the one of p_{k+1} (due to line 04). Once that happens, the if-statement conditions of line 04 cannot hold again for p_{k+2}.

The case of k<n-1 is showed by similar arguments to the ones in the case of k<n-2 except that $x_{n-1} = x_0$ cannot hold due to the proof of k=1.

Claim 1's proof end ☐

Next, we demonstrate super-self-stabilization. Let R be an execution of the algorithm, such that the starting configuration is safe, *i.e.*, $p_0,\ldots,p_{n-1}$ have each a colour that is unique within their individual neighbourhoods and these colours do not change (in the absence of transient faults and topological changes). Then, immediately after the system start, the environment takes a step that removes node p_k: 0<k<n. By line 07, p_{k+1} changes its colour to $\perp$ and thus there cannot be any further violation of the coloring task, i.e., the requirement of a safe passage to a safe configuration is satisfied. Moreover, within one asynchronous cycle, line 04 guarantees that all processors in the system have a legal colour assignment within a single asynchronous cycle. Thus, the optimal recovery requirement is satisfied (when starting in the safe configuration and then having a single change to the topology).

....

Question 5: **Self-stabilizing algorithm for finding the centers with the tree topology (12 points)**

Let G:=(V,E) be a graph, where V={p_1,...,p_n} is the set of vertices and E ⊆ V×V. This question considers only tree graphs. A tree is a graph with maximal amount of edges that does not contain a cycle. Let d(i, j) denotes the distance, *i.e.*, the length of a shortest path in G, between vertices p_i and p_j. Let e(i) = max{d(i, j) : p_j ∈ V} denote the eccentricity of a vertex p_i, which is the distance between p_i and the farthest vertex from p_i in G. Let center(G) = {p_i ∈ V : e(i) ≤ e(j), ∀p_j ∈ V} denote the set of centers of G, the set of vertices in V with minimum eccentricity. Jordan (1869) has proved that for trees, there are only two possibilities: (centered trees) exactly one center node, or (bicentered trees) exactly two center nodes that are adjacent.

```
Algorithm 1: Self-stabilizing algorithm for center-finding; code for $p_i$
1  Constants:
2  $N(i) = \{p_j : (p_i, p_j) \in E\}$ $p_i$'s neighbors;
3  Variables:
4  $h_i$ stores $p_i$'s $h$-value.
5  Macros:
6  $N_h(i) = \{h_j : (p_i, p_j) \in E\}$ $p_i$'s neighbors $h$-values (multiset);
7  $N_h^-(i) = N_h(i) \setminus \{\max(N_h(i))\}$ contains all elements of $N_h(i)$ without one
      maximum $h$-value. E.g., $N_h(i) = \{2, 3, 3\} \Rightarrow N_h^-(i) = \{2, 3\}$;
8  $leaf(i) = (|N(i)| = 1)$;
9  do forever
10 begin
11     if $(leaf(i) = true) \wedge (h_i \neq 0)$ then $h_i \leftarrow 0$;
12     if $(leaf(i) \neq true) \wedge (h_i \neq 1 + max(N_h^-(i)))$ then $h_i \leftarrow 1 +$ ______________;
```

5a. (1 point) Algorithm 1 presents a template for a self-stabilizing algorithm that finds the center(s) of a tree. The template is missing an expression in line 12. Please complete this expression.

It is simply $max(N_h^-(i))$

5b. (2 points) Please define the set of legal executions for Algorithm 1 (with the expression that you have completed for line 12).

We say that execution R of Algorithm 1 is legal if in any configuration the following two hold.

(i) Suppose that p_i is a leaf node in the tree. Then, h_i=0.

(ii) Suppose that p_i is not a leaf node in the tree. Then, h_i=1+ $max(N_h^-(i))$

5c. (6 points) Please demonstrate the convergence for Algorithm 1 (with the expression that you have completed for line 12).

We define the *height* of non-center node p_i as the height of the tree rooted at p_i after the removal of the edge that leads from p_i towards the any of the centers. Given a configuration *c*, we say that p_i's *height is floating* if, and only if, p_i's height is smaller than h_i's value.

Claim 2 implies the convergence property for Algorithm 1, because once it holds for any non-center node $p_{k''}$ that h_k=*height*$_k$, which denotes p_k's height, the center nodes p_k calculates h_k correctly to be one greater than $max(N_h^-(i))$ (due to line 12).

Claim 2 (convergence): within k+1 asynchronous cycles: (i) the minimum floating height in the system is at least k, and (ii) h_i equals to the height of every node p_i that its height is smaller than k.

Proof: This proof is by induction on p_k's height.

For the case of k=0, we note that h_0=0 holds within a single asynchronous cycle due to line 11.

Suppose that the claim is true for every k>0. We show that the claim is true for the case of k+1.

Let p_k be a node that none of its neighbours are center nodes. Moreover, let (p_k, $p_{k'}$) and (p_k, $p_{k''}$) be edges that leads towards, and respectively, away from the set of center nodes, such that the height of $p_{k''}$ is the highest among all such neighbours that are the least away from the set of center nodes. By the induction hypothesis, we know that *height*$_{k'}$ ≥ *height*$_{k''}$. Therefore in c_{k+1}, $(N_h^-(i))$ includes *height*$_{k''}$ and does not include *height*$_{k'}$ when *height*$_{k'}$ > *height*$_{k''}$, where c_k is the configuration the system reaches within k+1 asynchronous cycles. We argue that within one asynchronous cycle after c_{k+1}, the system reaches configuration c_{k+2} in which the claim's invariants (i) and (ii) hold with respect to k+1. In detail, since in c_{k+1} includes *height*$_{k''}$ and does not include *height*$_{k'}$ when *height*$_{k'}$ > *height*$_{k''}$, p_k calculates h_k correctly (due to line 12).

Claim 2's proof end ☐

5d. (1 point) Given Algorithm 1 (with the added expression in line 12, cf. item 5a), how can node p_i know whether it is part of the center set? Does your answer change in case where there is more than one node in the center set?

In a safe configuration, node p_i is part of the center set if, and only if, h_i is larger of equal to h_j, where p_j if any neighbor of p_i. This can be computed by reading the h-values of all neighbors. Note that this answer does not depends on the number of centers of the tree.

5e. (2 point) Please modify the code of Algorithm 1, if needed, so that every node knows the entire set of centers in the graph. Moreover, please state your assumptions clearly.

In a safe configuration, node p_i that is not part of the center set, can know which edge leads to the nearest node in the center set. It can simply look for a neighbor that its h-value is greater than its own.