Chalmers University of Technology

Computer Science and Engineering
Elad Michael Schiller

Written exam in EDA387/DIT671 Computer Networks 2022-10-27. Exam time: 4 hours.

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-speaking 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.
10. Do not hand in anything that is not part of the solution.
11. Always write your answers in your own words, using your own diagram, etc. In case you have to use the words of others, e.g., a formal definition, please make sure to use quotation marks and clearly identify the source.
12. Absolutely do not collaborate in any way or form with other people during the entire exam time (regardless of whether you are still writing the exam or not).

Question 1: true or false questions (24 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.

2a. (3 points) The exterior gateway protocol (EGP) uses Open Shortest Path First, aka OSPF (the link state algorithm) and Routing Information Protocol, aka RIP, (the distance vector algorithm) for making sure that all Autonomous Systems, aka AS, can selects and implement their own standards and policies.

False: OSPF and MIT are IGP protocols not EGP.

2b. (3 points) The protocol Routing Information Protocol, aka RIP, (the distance vector algorithm) was proven to be self-stabilizing by Radia Perlman. This protocol suffers from problems related to counting to infinity. That is, in the presence of failures, RIP is not guaranteed to recover within a period that is proportional to the network diameter All proposed remedies, e.g., split horizon and poison reverse, do not completely solve the problem. In fact, there is no known algorithm, self-stabilizing or not, that can overcome this challenge and regain consistency within the time that is proportional to the network diameter.

False: we have studied in class we studied a self-stabilizing that solve the problem in time that is proportional to the network diameter.

2c. (3 points) IPv4 uses 32-bit address space. IPv6 uses 128-bit address space. If a 128-bit address space was used already in IPv1, there would be no need to invent a number of protocols (or they could have had a much simpler implementation). These protocols include Address Resolution Protocol (ARP), Network address translation (NAT), and Dynamic Host Configuration Protocol (DHCP).

True

2d. (3 points) There is no self-stabilizing algorithm for leader election in anonymous networks in which the processors do not have globally unique identifiers. When the processors have globally unique identifiers, there is a self-stabilizing leader election algorithm that stabilizes within O(n) and uses messages of size O(log N) bits. There is also a self-stabilizing leader election algorithm that stabilizes within O(d) and uses messages of size O(n log N) bits. Recall that we denote the number of processors by n and use N to denote an upper bound on n. Also, d is the network diameter.

True

2e. (3 points) There is no self-stabilizing deterministic link layer protocol that uses a bounded local memory and bounded message size in systems in which the channel capacity is unbounded. One way to overcome this impossibility is to use the alternating bit protocol in which the sender uses a random bit, of either 0 or 1, whenever it fetches a new message from the application layer.

False. The random solution uses three values, 0, 1, and 2.

2f. (3 points) In class we studied a self-stabilizing algorithm for clock synchronization that can deal with napping faults. This algorithm cannot deal with crash failures in which processors stop taking steps forever.

False. Napping faults can model crash faults.

2g. (3 points) Network Time Protocol (NTP) is a clock synchronization protocol that uses a constant number of bits in its clock variables and fields. This contradicts the lower bound studied in class that says that there is no clock synchronization protocol that uses a constant number of bits. This is an example of the difference between theory and practice.

False. NTP is a hierarchical solution whereas the ones studied in class are peer-to-peer solutions.

2h. (3 points) Self-stabilizing systems always recover within a finite time after the last occurrence of a transient fault, which may bring the system to an arbitrary configuration. Also, super-self-stabilizing systems always recover eventually after the last occurrence of a transient fault, which may bring the system to an arbitrary configuration. Moreover, these systems offer a quick provision of safe passage guarantees. Furthermore, during legal executions, the recovery from any single topology change occurs within a constant time. Pseudo-self-stabilizing systems do not guarantee recovery within a finite time after the last occurrence of a transient fault. Instead, they offer a bounded number of deviations from the requirements of the abstract task.

True.

.....

Question 2: Self-stabilizing colouring in uniform anonymous networks (20 points)

This question considers the design of a deterministic self-stabilizing algorithm for vertex coloring in uniform anonymous networks. Uniform anonymous networks do not have distinguished nodes and all nodes run identical programs without access to globally unique node identifiers. The problem vertex coloring requires the assignment of an integer, called color, to every node in the network. These colors are selected from a predefined set, named palette. In this question, we assume that the palette includes at least $\Delta+1$ colors, and thus, selecting an optimal number of colors is not relevant.

2a (2 points) Consider the case in which the network topology is a general graph. Show that there is no self-stabilizing deterministic algorithm for vertex coloring in general graphs. Please include all the details in your proof.

The solution is similar to the ones presented in class regarding leader election and token circulation.

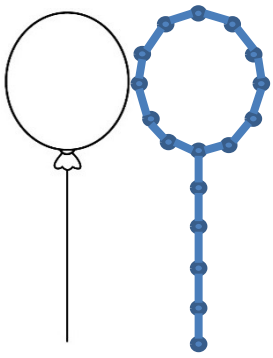


Figure 1 Balloon graph $B_{12,6}$

The rest of this question consider communication networks with the topology of a balloon graph. A balloon graph $B_{x,y}$ =(V,E) is constructed using an undirected ring (circle) graph C_x =(V_x,E_x) and chain (line) graph L_y =(V_y,E_y), where V= $V_x \cup V_y$ and E= $E_x \cup E_y$. This construction assumes that $V_x \cap V_y = \{v_{pivot}\}$, where x and y are unknown integers. Figure 1 illustrates a balloon next to the balloon graph $B_{12,6}$.

2b (5 points) Please design a self-stabilizing deterministic algorithm for vertex coloring in balloon graphs. Please provide a clearly written pseudo code of your algorithm. Please make sure that your algorithm does not assume that x and y are known. Your solution is allowed to use a combination of algorithms that we learned in class as long as your clearly name them and clearly explain all the elements of your solution. Please define all the variables in use and list all the needed assumptions.

Use the self-stabilizing ranking algorithm for transforming the algorithms to be one with identifiers. Use the node with only one neighbor as a distinguished node. Then, use the self-stabilizing coloring algorithm.

2c (1 points) Please define a safe configuration for your solution.

Identical to the definition studied in class.

2d (4 points) Please show that when starting in a safe configuration, your solution does not change the colors associated with each node. Prove your claim clearly.

Via the fair composition theorem and the properties of the ranking and coloring algorithms.

2e (5 points) Please show that when starting in any configuration, your solution reaches a safe configuration eventually. Prove your claim clearly.

Via the fair composition theorem and the properties of the ranking and coloring algorithms.

2f (2 points) Please show a bound on the stabilization time. Prove your claim clearly.

O(n), via the fair composition theorem and the properties of the ranking and coloring algorithms.

2g (1 points) Is there a contradiction between your proof in item 2a and your proposed solution? Explain your claims clearly.

No, because 2a considers a general graph rather than a balloon graph.

Question 3: Self-stabilizing discrete pursuit-evasion network searching (12 points)

Discrete pursuit-evasion network searching is a problem in computer networks in which one group attempts to track down members of another group in the network. Specifically, consider two finite sets of agents, named pursuers and evaders, that occupy nodes in the networks. The two sides take simultaneous steps, which consist of each member either staying put or moving along an edge to an adjacent node. If a pursuer occupies the same node as an evader the evader is captured and removed from the network. This question considers different numbers, k, of pursuers that are necessary to ensure the eventual capture of all the evaders, where the number of the latter is finite yet unbounded. The number, k, of pursuers can depend on the network properties.
We assume that the evaders have unlimited ability to know the pursuers' whereabouts and coordinate their actions as well as their strategy. Your mission is to design the strategy of the pursuers as well as the communication protocol that facilitates the pursuers' strategy. Note the system does not provide the pursuers a global picture of the agents' whereabouts. However, any node in which an agent resides receives a local indication about the agent's presence and its type, i.e., pursuerPresent, evaderPresent ∈ {true, false}.

One may assume that the pursuers and evaders cannot travel simultaneously on the same edge without getting caught.

3a. (2 points) Please provide a civil application to the studied problem of discrete pursuit-evasion network searching. Please explain what the benefits are in modeling the proposed application using the studied problem.

There are many relevant applications. The idea is that the model allows you to chartreuse the conditions for the solution's existence as well as design algorithms within bounds that have guaranteed optimality.

3b. (1 points) Please show that for the case of k=1, there are networks for which the pursuers cannot guarantee the removal of all evaders---even if there is just one evader in the network. Prove your claim clearly.

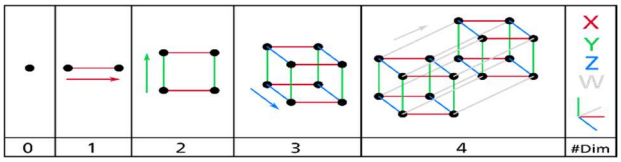


Figure 2 Constructing hypercube of dimensions x=0,1,…,4.

Consider the directed ring network and assume that the evader knows the starting state and the algorithm of the pursuer. Then, show that the pursuer never catches the evader.

3c. (1 points) Please show that for the case of k=1 and the network of the chain topology of n node, there is a strategy for which the pursuer guarantees the removal of all evaders. How many steps does the pursuer needs to take until all evaders are removed? Prove your claim clearly.

Move to one side of the chain and then to the other. Show that the pursuer never catches the evader within O(n) steps.

The rest of the question considers networks that have the topology of a hypercube. A hypercube can be defined by increasing the number of dimensions of a shape. Figure 2 illustrates how this construction occurs:

The 0-dimension hypercube is a single node.

The 1-dimension hypercube is a graph with two nodes and one connecting edge.

The 2-dimension hypercube can be constructed by taking two 1-dimension hypercubes and using 2 edges for connecting them.

The 3-dimension hypercube can be constructed by taking two 2-dimension hypercubes and using 4 edges for connecting them.

The 4-dimension hypercube can be constructed by taking two 3-dimension hypercubes and using 8 edges for connecting them.

More generally, the (d+1)-dimension hypercube can be constructed by taking two d-dimension hypercubes and using 2^d edges for connecting them.

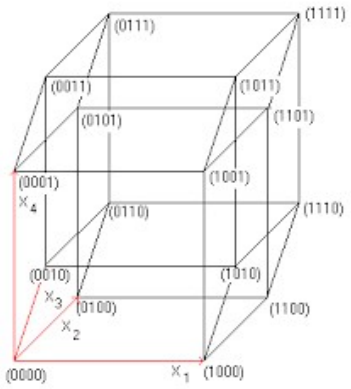


Figure 3 Addresses in hypercubes

Figure 3 illustrates how to address nodes in hypercubes.

3d. (2 points) Find a strategy for the removal of an unbounded number of evaders using 2^k pursuers on a (k+1)-dimension hypercube. How many steps does the pursuer needs to take until all evaders are removed?

Assume that each pursuer has a unique identifier that is between 0 and 2^k-1. Consider the two subgraphs of the k-dimension hypercubes. Give them address like in Figure 3. Notice that they have an address that differ by one entry in the vector of addresses. Thus, we can name these two k-dimension hypercubes as 0 and 1. The strategy has two phases, 0 and 1. Moreover, each pursuers can use its identifier for having a unique node at each k-dimension hypercube. Also, when an agents move from the node in hypercube 0 to hypercubes 1 (or via versa), it needs to travel over one edge. At phase x, all hypercubes place themselves at their unique node at hypercube x.

3e. (3 points) Please design a self-stabilizing communication protocol for coordinating the action of the k pursuers according to the strategy in Item 3d. You may assume that the network is synchronous.

You need a self-stabilizing clock synchronization algorithm, say, converge to the min with 2 (k+1) clock states. Use this clock for generating the phases of 0 and 1. The self-stabilizing algorithm makes each pursuer to move to its location according to the phase.

3f. (3 points) Please prove the correctness of the solution you have proposed in Item 3e.

Question 4: Self-stabilizing all pairs shortest path discovery in semi-anonymous networks (8 points)

Semi-anonymous networks assume that nodes do not have access to globally unique identifiers. Also, the same program is run by all nodes except one, which is called the distinguished (or root) node since it runs another program. The problem of All Pairs Shortest Path (APSP) requires the calculation of the shortest path between any pair of nodes. There are several (non-distributed) sequential algorithms that solve the APSP problem, such as the Floyd–Warshall algorithm, Pettie-Ramachandran 2002, Williams 2014, and Seidel 1992. Your task is to provide a (distributed) self-stabilizing APSP algorithm for semi-anonymous networks.

4a. (4 points) Please explain how one can overcome the fact that there are no globally unique identifiers and still offer a meaningful output. Your solution is allowed to use a combination of algorithms that we learned in class as long as your clearly name them and clearly explain all the elements of your solution.

Use the studied self-stabilizing ranking.

4b. (4 points) Please explain how the fact that there are known sequential algorithms can help to offer a self-stabilizing solution. Your solution is allowed to use a combination of algorithms that we learned in class as long as your clearly name them and clearly explain all the elements of your solution.

Use the studied self-stabilizing transformation from sequential to distributed.

Appendix

Selected algorithms learned in class.

```
01  P_1:        do forever
02                    if x_1=x_n then x_1 := (x_1+1) mod (n+1)
03  P_i (i ≠ 1): do forever
04                    if x_i ≠ x_i-1 then x_i := x_i-1
```

```
Program for p_i :
01 do forever
02      if pointer_i = null and (∃ p_j ∈ N(i) | pointer_j = i ) then
03              pointer_i = j
04      if pointer_i = null and (∀ p_j ∈ N(i) | pointer_j ≠ i ) and
05                      (∃ p_j ∈ N(i) | pointer_j = null ) then
06              pointer_i = j
07       if pointer_i = j and pointer_j = k and k ≠ i then
08              pointer_i = null
09 od
```

```
01 do forever
02      ⟨candidate,distance⟩ := ⟨ID(i), 0⟩
03      forall P_j ∈ N(i) do
04         begin
05              ⟨leader_i[j],dis_i[j]⟩ := read⟨ leader_j, dis_j ⟩
06              if (dis_i[j] < N) and ((leader_i[j] < candidate) or
07                 ((leader_i[j] = candidate) and (dis_i[j] < distance))) then
08                      ⟨candidate,distance⟩ := ⟨ leader_i[j],dis_i[j] + 1⟩
09         end
10      write⟨leader_i ,dis_i⟩ := ⟨candidate,distance⟩
11 od
```

```
01      initialization
02      begin
03         i := 1
04         bit_s := 0
05         send(⟨bit_s,im_i⟩) (*im_i is fetched*)
06      end (*end initialization*)
07      upon a timeout
08              send(⟨bit_s,im_i⟩)
09      upon frame arrival
10      begin
11              receive(FrameBit)
12              if FrameBit = bit_s then (*acknowledge arrives*)
13                 begin
14                      bit_s := (bit_s + 1) mod 2
15                      i := i + 1
16              end
17               send(⟨bit_s,im_i⟩) (*im_i is fetched*)
18      end
```

```
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_mi := read(r_mi)
06              FirstFound := false
07              dist := 1 + min{ lr_mi.dis | 1 ≤ m ≤ δ }
08              for m := 1 to δ
09              do
10                      if not FirstFound and lr_mi.dis = dist -1
11                              write r_im := ⟨1,dist⟩
12                               FirstFound := true
13                      else
14                              write r_im := ⟨0,dist⟩
15              od
16      od
```

```
01 Root: do forever
02              lr_1,i := read (r_1,i)
03              if lr_δ,i = r_i,1 then
04                      (* critical section*)
05                       write r_i,2 := (lr_1,i + 1 mod (4n -5))
06              for m := 2 to δ do
07                      lr_m,i = read(r_m,i)
08                      write r_i,m+1 := lr_m,i
09              od
10      od
11 Other: do forever
12              lr_1,i := read (r_1,i)
13              if lr_1,i ≠ r_i,2
14                      (* critical section*)
15                       write r_i,2 := lr_1,i
16              for m := 2 to δ do
17                              lr_m,i := read (r_m,i)
18                              write r_i,m+1 := lr_m,i
19              od
20      od
```

```
01      initialization
02      begin
03         j := 1
04         bit_r := 1
05      end (*end initialization*)
06      upon frame arrival
07      begin
08              receive(⟨FrameBit , msg⟩)
09              if FrameBit ≠ bit_r then (*a new message arrived*)
10                 begin
11                      bit_r := FrameBit
12                      j := j + 1
13                      om_j := msg (*om_j is delivered*)
14              end
15               send(bit_r)
16      end
```

```
01 Root: do forever
02                  ID_i:= 1
03                  sum := 0
04                  forall p_j ∈ children(i) do
05                          lr_ji := read(r_ji)
05                          write r_ij.identifier := Id + 1 + sum
06                          sum := sum + lr_ji.count
08                  od
09          od
10 Other: do forever
11                  sum:= 0
12                  lr_parent,i := read(r_parent,I)
13                  ID_i := lr_parent,I.identifier
14                  forall p_j ∈ children(i) do
15                          lr_ji := read(r_ji)
16                          write r_ij.identifier := Id_i + 1 + sum
17                          sum := sum + lr_ji.count
18                  od
19          od
01 upon a pulse
02          forall P_j ∈ N(i) do send (j, clock_i)
03          max := clock_i
04          forall P_j ∈ N(i) do
05                  receive(clock_j)
06                  if clock_j > max then max := clock_j
07          od
08          clock_i := (max + 1) mod ((n +1)d +1)
```

```
01 do forever
02  Readset_i :=∅
03  forall p_j∈N(i) do
04      Readset_i := Readset_i ∪ read(Processors_j)
05  Readset_i := Readset_i \ <i,*> where \ is the set subtraction symbol
06  Readset_i := Readset_i ++ <*,1>
07  Readset_i := Readset_i ∪ {<i,0>}
08  forall p_j ∈ processors(Readset_i) do
09      Readset_i := Readset_i \ NotMinDist(P_j, Readset_i)
10  write Processors_i := ConPrefix(Readset_i)
11od
```

The program for P:

1) Read every count and clock
2) Find the processor set R that are not behind any other processor
3) If R ≠ ∅ 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)

```
01 Root: do forever
02          sum:= 0
03                  forall p_j ∈ children(i) do
04                          lr_ji := read(r_ji)
05                          sum := sum + lr_ji.count
06                  od
07                  count_i = sum + 1
08          od
09 Other: do forever
10                  sum:= 0
11                  forall p_j ∈ children(i) do
12                          lr_ji := read(r_ji)
13                          sum := sum + lr_ji.count
14                  od
15                  count_i = sum + 1
16                  write r_i,parent.count := count_i
17          od
01 upon a pulse
02          forall P_j ∈ N(i) do send (j,clock_i)
03          min := clock_i
04          forall P_j ∈ N(i) do
05                  receive(clock_j)
06                  if clock_j < min then min := clock_j
07          od
08          clock_i := (min + 1) mod (2d +1)
```

```
01  Do forever
02          GColors := 0
03          For m:=1 to δ do
04                  lr_m:=read(r_m)
05                  If ID(m)>i then
06                     GColors := GColors U {lr_m.color}
07          od
08          If color_i ∈ GColors then
09            color_i:=choose(\\ GColors)
10          Write r_i.color := color_i
11  od
```

```
01  Do forever
02          AColors := {}
03          GColors := {}
04          For m:=1 to δ do
05                  lr_m:=read(r_m)
06                  AColors:=AColors U {lr_m.color}
07                  If ID(m)>i then GColors := GColors U {lr_m.color}
08          od
09          If color_i = ⊥ or color_i ∈ GColors then
10                   color_i:=choose(\\ AColors)
11          Write r_i.color := color
12  od
13  Interrupt section
14          If recover_ij and j > i then
15                  Color_i := ⊥
16                  Write r_i.color := ⊥
```