CHALMERS
EXAMINATION / TENTAMEN

Course code/kurskod	Course name/kursnamn			
EDA 387	Computer Networks			
Anonymous code Anonym kod		Examination date Tentamensdatum	Number of pages Antal blad	Grade Betyg
EDA387-0002-HBZ		25/8 - 22	10	4

* I confirm that I've no mobile or other similar electronic equipment available during the examination.
Jag intygar att jag inte har mobiltelefon eller annan liknande elektronisk utrustning tillgänglig under eximinationen.

Solved task Behandlade uppgifter No/nr		Points per task Poäng på uppgiften	Observe: Areas with bold contour are to completed by the teacher. Anmärkning: Rutor inom bred kontur ifylles av lärare.
1	X	7	
2	X	9	
3	X	5	
4	X	6½	
5	X	10	
6	X	10	
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
Bonus poäng			
Total examination points Summa poäng på tentamen		42	(41½)

4

a) The purpose of the algorithm is to create a first BFS tree. This is the basis for many other algorithms I Beliave that the algorithm is simply called "Distributed Spanning tree protocol" It is used, among other things, to construct a logical network topology without loops.

b) The functionality of the packet forwarding rules are that if a packet matches one of the rules it will be sent directly to the location specified by the rule by the switch without involvement of the SDN controller.

The Distributed spanning tree protocol could probably be used to some extent. The constructed tree is the basis for some routing algorithms (i think may be the shortest path first algorithm)

However since the code would be run on the SDN controller which is centralized and has knowledge of the network topology directly. It would probably be better to use another centralized solution.

c) IPv4 allows for $2^{32} \approx 4$ billion unique adresses, This was more than enough when IPv4 was first created. However with the enormous growth of the internet, we are running out of IPv4 adresses. This results in restrictions on the size of the internet. IPv4 adresses become increasingly harder to aquire and could possibly become expensive. IPv6 increases the address size to 128 bits this results in a very large amount of possible adresses solving the size limit of the internet as given by IPv4.

IPv6 also diminishes the need for protocols such as Network Adress translation (NAT) which is used to allow several computers to share an outbound IP-adress in order to conserve IP Adresses. It could also simplify Adress resolution protocol (ARP) as adresses could be made short

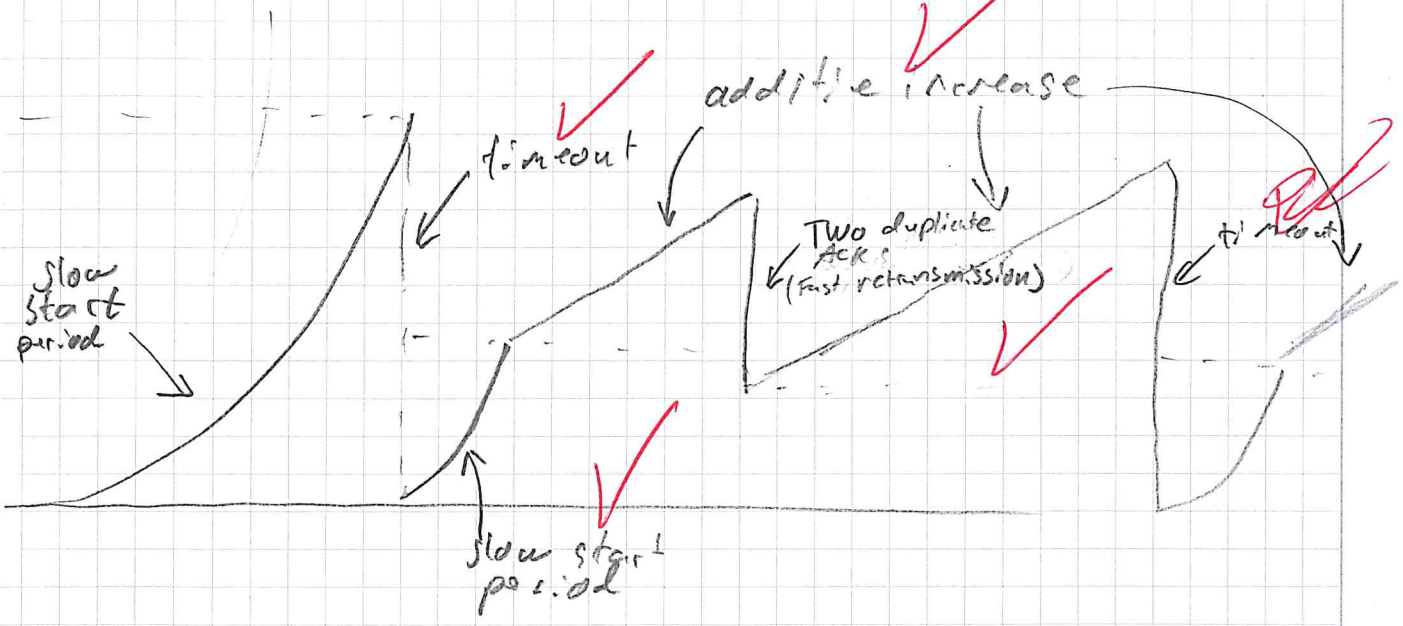
4

d) (1) The proposed IP address would have an impractically large overhead, probably hindering the adoption of IP and causing whatever replacement internet to not use IP as a base. The main issue is the large overhead, 74819-bits allows for an absurdly large number of addresses, it would guess, probably many orders of magnitude more than would be needed to give a unique adress to every atom (likely every elementary particle) in the entire universe. This means that there would be absolutely no point in having such a large adress.

If this protocol was somehow adopted there would obviously be no real limit to the size of the internet. There would be no need for NAT as explained in c) and likely no need for AIRP as MAC adresses could be directly encoded in the IP adress

d)(2) e) Super self stabilizing systems are better than self stabilizing systems because, while both self stabilizing and super self stabilizing systems guarantee recovery from an arbitrary state in a bounded time, the time needed to do so will be constant (of order $O(1)$) in any system configuration for super self stabilizing systems, while the time could be of orders such as $O(n)$, $O(n^2)$ or $O(d)$ (depending on the system configuration) in regular self stabilizing systems.

We should however not only focus on super self stabilizing systems since there are many problems for which a self stabilizing solution exists but no super self stabilizing solution exists.


timeout
additive increase
Two duplicate Acks (Fast retransmission)
timeout
Slow start period
Slow start period

4

a) Agreement: Identical clock values 4/4

Progress: The clock values are increasing every pulse.

b) The algorithm assumes unbounded variables. this can not be implemented in practice.
Problems arise when the clock rolls over and becomes 0. Having large variable sizes does not help since the state could be corrupted in a way causing the clock variable to be close to the point where it will roll over. 1/2

c) Assuming no transient faults and unbounded variables:

Induction!

After 0 pulses, every processor has the maximum value of every processor 0 steps away

True, The processors have their own clock values. They themselves are the only processor at distance of 0.

Assuming every processor has the maximum clock value of every processor at distance n-1 or less after n-1 pulses
After n pulses, every processor has the maximum value of every processor at distance n or less. 3/3

proof: an arbitrary processor P_i, its neighbors has the value of every processor at a distance of n-1 or less, the distance to any of the neighbors is 1. P_i will assume the maximum clock value of any of its neighbors. Therefore P_i will have the maximum clock value of every processor of distance n or less after n pulses.

Due to this after d pulses, every processor will have the maximum clock value of every processor at distance d or less, which is every processor since d is the graph diameter.
As such, the processors will be synchronized

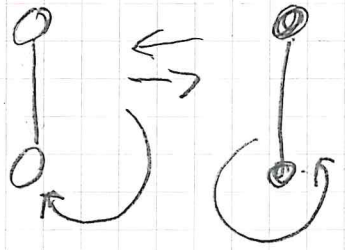
4

where is a2

b) I will assume that the processors either have unique identifiers or that there is one distinguished processor

In the case of a distinguished processor this is converted to an id based system using fair composition with the spanning tree - routing - naming composition described in the course as such, I will now assume an id based system

If no ids exist and a central daemon triggers every processor to take a step simultaneously we run in to this issue,



where as shown in the illustration two processors linked will not be able to break the symmetry

My proposed algorithm is as follows

P_i do forever

```
free := true
for all neighbours Pj
    if (j > i) and Pj.in_set
        free := false
end
if free
    in_set = true
else
    in_set = false
send value of in_set to all neighbors
end
```

in short:
if no neighbor with higher id is in the set, P_i joins the set
else P_i leaves the set

4

4

c) The set of safe configurations for my algorithm is as follows:

(1) There are no neighbors that are both in these

(2) if none of processor P_i:s neighbors are in the s... then P_i is in the set

(3) Processors with higher id have higher priority to be in the set. P_n will allways be in the set, the rest is stabilized from there

d) proof of (1) if there are two neighbors both in the set, the processor with lower id will notice it has a neighbor with higher priority in the set and leave the set.

(2) if none of a processors neighbors are in the set, the processor will join the set.

(3) the processors with higher ids will not pay attention to the processors with lower ids

1 1 2

It is possible that the algorithm will leave a maximal independant set if the priorities are not correct according to (3) but this is not a safe state. Once the system has stabilized, nothing should change

4

a) There should be exactly one token at any given time.

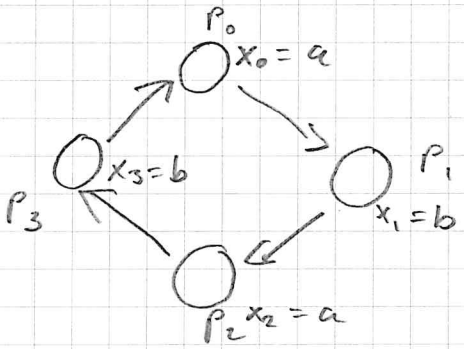
In a fair execution, every processor is to get the token within a bounded time.

In a ring this usually means that the token is to be given to the next processor in the ring after every pulse.

In Dijkstras token circulation algorithm, P_0 has the token if X_0 and X_{n-1} are equal, P_i $(i \neq 0)$ has the token if $X_i \neq X_{i-1}$

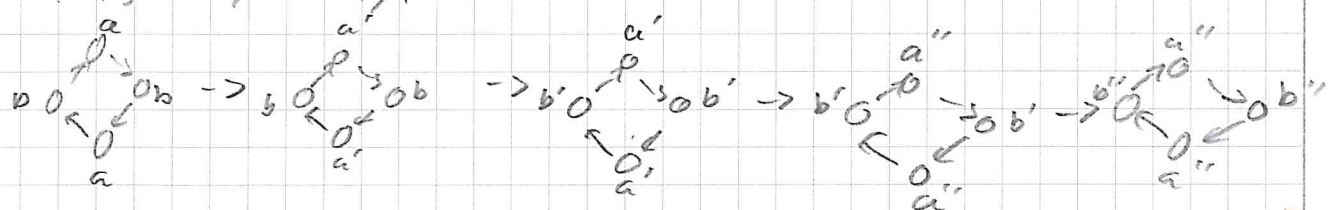
Sorry, i have realized i have called P_1 as P_0 and P_n as P_{n-1}, the explanations should still be correct however

b) consider this case



consider the execution order $P_0, P_2, P_1, P_3, P_0, P_2, P_1, P_3$.

Since P_0 and P_2 both have $x = a$ and can only see $x = b$ in the previous processor, they will get the same value after the pulse. A similar argument gives that P_1 and P_3 will now see the same values and as such get the same values, this will continue since all processors run the same algorithm, there is no way to break this symmetry.



4

c) Lemma 2.2 states that a configuration in which all x variables are equal is safe.

This is true since, in this configuration only one token exists. P_0 has this token, the next tick P_0 will have a different value, as such the token will be given to P_1, since P_1 will then adopt the new value of P_0, the token will be given to P_2. This will continue until the new value reaches P_{n-1} then the cycle will repeat, this means that the algorithm is working as intended.

Lemma 2.4 states that P_0 (what you call P_1) changes its value at least once every n rounds

This is true since, if the current value of P_0 does not exist in any other processor, it will take $n-1$ rounds for the value to reach P_{n-1}, the next round, P_0 will change its value

If the value of P_0 already exists in some x_i, $i \neq 0$ it will take less than $n-1$ rounds for this value to reach P_{i-1} and cause P_0 to change the next round

both these lemmas show how the value of P_0 propagates to other processors

Lemma 2.2 does not really use this, it can be proved by showing that the configuration where all x_i are the same is safe, this is easy to prove as you only need to show that there exists exactly one token and that this token will propagate properly.

Lemma 2.4 uses the propagation to give a bound on the time it takes for P_0 to introduce the next value in the sequence.
of x_0

4

4

a) Every processor has a boolean variable m_i encoding if it is matched with the previous processor

P_0: do forever
$m_0 := false$
end

P_i : do forever
(i ≠ 0)
if $m_{i-1} = false$ then
$m_i := true$

else
$m_i := false$

end

b) The legal set of executions is that in which no two concurrent processors in the ring are both or more matched to the previous processor.

In addition, since this is maximum matching, there must be at most one unmatched processor if there is an odd number of processors in the ring and NO unmatched processors if there is an even number of processors in the ring.

By unmatched processor i mean a processor P_i where $m_i = false$ and $m_{i+1 (mod\ n)} = false$

c) My algorithm will stabilize in order O(n), this is because a processor will assume their final state the pulse immediately after the processor before it reaches its final state. Since P_0 will assume its final state after the first pulse, P_1 after the second and so forth until P_{n-1} assumes its final state after n pulses. The reason P_i assumes its final state once P_{i-1} assumes its final state is that P_i only uses the state of P_{i-1} to compute its state

Once the algorithm has stabilized, the states of the m variables of the processors will be false, true, false, true - - - where $m_i = false$ if i is even and $m_i = true$ if i is odd

Answer only one question on this page. Do not write on the back of this paper
Behandla endast en uppgift på detta blad. Skriv ej på baksidan

c) and d) cont. If there is an even number of processors in the ring, the index of the last processor will be odd (the last index will be $n-1$, n is even)

$(P_0\ P_1)\ (P_2\ P_3)\ \ldots\ (P_{n-2}\ P_{n-1})$

this means that the last processor will be have m_{n-1} = true, as such the last processor will be matched, since the sequence will be

(false, true, false, true) ... (false, true)

no unmatched processors will exist (Optimality)

If there are an odd number of processors

m_{n-1} will be false. The sequence will be

(false, true, false, true) ... (false, true, false)

and only one unmatched processor (P_{n-1}) will exist (Optimality)

Therefore, my algorithm is self stabilizing and optimal after convergence.

The state machine of my algorithm needs only two states, if the processor is or is not matched to the previous processor. This is the variable m_i