ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Autumn Term 2021 Prof. R. Wattenhofer

Computer Systems – Part 2
Exam

Date: 31-01-2022

Do not open or turn before the exam starts!
Read the following instructions!

The exam takes 90 minutes and there is a total of 90 points. The maximum number of points for each subtask is indicated in brackets. **Justify all your answers** unless the task explicitly states otherwise. Mark drawings precisely.

Answers which we cannot read are not awarded any points!

At the beginning, fill in your name and student number in the corresponding fields below. You should fill in your answers in the spaces provided on the exam. If you need more space, we will provide extra paper for this. Please label each extra sheet with your name and student number.

Family Name	**First Name**	**Student Number**

Task	**Achieved Points**	**Maximum Points**
1 - Choose the Correct Answer		18
2 - Paxos		18
3 - Agreement		20
4 - Bitcoin		18
5 - Game Theory		16
Total		**90**

1 Choose the Correct Answer (18 points)

Each question has **exactly one correct** answer. Indicate your answer by checking the corresponding circle. A correct answer gives **2 points**, a wrong answer gives **0 points**, and no answer gives **0 points**. You do **not** need to explain your answers.

a) To achieve the most exact synchronization, NTP most importantly requires:

- ○ Low network delay times
- ○ Stable network delay times between multiple calls
- ○ Symmetric network delay times
- ○ Many iterations until it converges

b) In Bitcoin, running a full-node does **not** let the operator:

- ○ Broadcast their transactions to the Bitcoin network.
- ○ Get their share of the block reward (minus fees).
- ○ Verify whether their own transactions got confirmed or not.
- ○ Help other nodes "catch-up" from scratch to the current state of the blockchain.

c) Regarding Bitcoin's UTXO set size and the blockchain size:

- ○ Both increase monotonically.
- ○ UTXO set stays static and the blockchain size increases monotonically.
- ○ UTXO set size increases monotonically, and the blockchain size could in theory go down to zero.
- ○ UTXO set size changes over time, and the blockchain size increases monotonically.

d) In Bitcoin, a selfish miner starts mining the next block on top of a previous block without broadcasting the previous block to the network. What information are they trying to hide?

- ○ The block-hash of the newly mined block.
- ○ Their share of the total network hash-power.
- ○ The confirmed transactions of the newly mined block.
- ○ The nonce that allowed them to solve the Proof of Work function.

e) A quorum system is defined over 100 servers. Which of the following might be true?

- ○ The load is 0.01.
- ○ The work is 0.5.
- ○ The system is 4-masking and there is a quorum Q in the system with $|Q| = 8$.
- ○ None of the above.

f) In PBFT, a correct backup b has executed a request r with sequence number s in view v. Which of the following is guaranteed?

- ○ No node executes a request $r' \neq r$ with sequence number s.
- ○ No node can create a prepared-certificate for (v, s, r') with $r' \neq r$.
- ○ At the time when b executed r, every correct backup was in view v.
- ○ Every correct backup eventually pre-prepares r for (v, s).

g) Which statement about an algorithm satisfying all-same validity is **wrong**?

- ○ If no correct node starts with the input v, the decision value cannot be v.
- ○ Correct nodes can decide on a value that was also proposed by byzantine nodes.
- ○ If all nodes start with the same input v, the decision value must be v.
- ○ None of the above.

h) Regarding consensus, which statement is true?

- ○ A single crash can prevent any deterministic algorithm from reaching consensus in the asynchronous model.
- ○ A single byzantine node can prevent any deterministic algorithm from reaching consensus in the asynchronous model.
- ○ No algorithm satisfies byzantine agreement with $n = 3f$.
- ○ All of the above.

i) In a scenario where 12 nodes act as client and server simultanously during an execution of the Paxos algorithm, which of the following is true?

- ○ Paxos tolerates the crash of at most 6 nodes.
- ○ Paxos tolerates the crash of at most 11 nodes.
- ○ Paxos tolerates 1 byzantine node.
- ○ None of the above.

This page is intentionally left blank.

Solutions

a) (3) NTP only measures the total network RTT and for synchronization assumes that half of that is the network delay in each direction. Therefore, the more symmetric this delay is, the more accurate the synchronization.

b) (2) Running a full node doesn't give any block rewards to the operator.

c) (4) The answer is "UTXO set size changes over time, and the blockchain size increases monotonically". Varying numbers of UTXO's are consumed and produced in any single transaction, and the set grows or shrinks based on such usage. The blockchain stores all historical transactions and hence keeps growing in size.

d) (1) The block-hash of the newly mined block. The selfish miner wants to be the only one mining on top of that block - so that they have a chance to take the lead.

e) (4) None of the above.

The theorems in "Chapter 21: Quorum Systems" help us. By Theorem 21.6 the load of the system is at least $1/\sqrt{n} = 1/10 = 0.1$, so the first option is incorrect. Work is defined as the expected size of the chosen quorum, with respect to the access strategy. Since all quorums have size at least 1, this value is also at least 1, ruling out the second option. By definition, in a 4-masking system the intersection of any two quorums has size at least $2 \cdot 4 + 1 = 9$, so all quorums need to have sizes at least 9, also ruling out the third option.

f) (2) No node can create a prepared-certificate for (v, s, r') with $r' \neq r$.

The first option is incorrect: faulty nodes can choose to execute different requests.

The second option follows from Lemma 26.18 in "Chapter 26: Authenticated Agreement". As backup b executed request r with sequence number s in view v, at least one correct backup has created a prepared-certificate for (v, s, r). Then, using Lemma 26.18, we obtain that no node can create a prepared-certificate for (v, s, r') with $r' \neq r$.

The third option is incorrect: it is possible that a correct backup is slow and is in view $v' < v$.

The fourth option is incorrect: the primary is not guaranteed to be correct. Hence, some correct backups may never receive the required pre-prepare message from the primary.

g) (1) The first option does not hold true, and is thus the right answer. Indeed, if it's not the case that all correct nodes start with the same value, then all-same validity does not restrict the decision value at all. Thus, option two is true and so is option three, as 'all nodes' includes 'all correct nodes', and thus if all nodes start with the same value, all-same validity dictates the decision value.

h) (4) Theorem 16.14 proves that the first two statements hold true. The third statement holds true by Theorem 17.13. Thus all statements are correct.

i) (4) Paxos tolerates less than $\frac{n}{2}$ crash failures, which means at most 5 in our example. Thus the first two statements are false. Paxos does not tolerate a single byzantine node, which means no statement holds true.

2 Paxos (18 points)

a) [6] What consensus properties does Paxos satisfy? Devise an "Anti-Paxos" consensus algorithm for binary inputs which satisfies the **opposite** consensus properties to Paxos (satisfies the properties that Paxos does not satisfy, and does not satisfy the properties that Paxos does satisfy).

Algorithm 1 Anti-Paxos

$v_i \in \{0, 1\}$ ◁ node's input bit

return v_i ◁ final value chosen by the node

From now on, let us consider the "Changed Paxos" algorithm below, where line 8 was changed from

$$\text{Pick } (T_{\text{store}}, C) \text{ with largest } T_{\text{store}}$$

to

$$\text{Pick } (T_{\text{store}}, C) \text{ with smallest } T_{\text{store}} .$$

Algorithm 2 Changed Paxos

	Client (Proposer)	**Server (Acceptor)**
	Initialization	
	c ◁ *command to execute* $t = 0$ ◁ *ticket number to try*	$T_{\max} = 0$ ◁ *largest issued ticket* $C = \perp$ ◁ *stored command* $T_{\text{store}} = 0$ ◁ *ticket used to store* C
	Phase 1	
1:	$t = t + 1$	
2:	Ask all servers for ticket t	
3:		**if** $t > T_{\max}$ **then**
4:		$T_{\max} = t$
5:		Answer with `ok`(T_{store}, C)
6:		**end if**
	Phase 2	
7:	**if** a majority answers `ok` **then**	
8:	Pick (T_{store}, C) with smallest T_{store}	
9:	**if** $T_{\text{store}} > 0$ **then**	
10:	$c = C$	
11:	**end if**	
12:	Send `propose`(t, c) to same majority	
13:	**end if**	
14:		**if** $t = T_{\max}$ **then**
15:		$C = c$
16:		$T_{\text{store}} = t$
17:		Answer `success`
18:		**end if**
	Phase 3	
19:	**if** a majority answers `success` **then**	
20:	Send `execute`(c) to every server	
21:	**end if**	

b) [8] This version of the algorithm does not always work. Assume that we have three clients (A, B, C) that want to execute their respective commands (a, b, c) on three servers ($S1$, $S2$, $S3$). Provide a sequence of events that leads to servers failing agreement.

c) [4] Fix the pseudocode of Changed Paxos, without changing line 8, such that Changed Paxos works properly. This only requires minor changes. You can write the changes on top of the pseudocode above.

a) Paxos does not achieve termination, but achieves validity and agreement. So, we need an algorithm, which achieves termination, but does not always achieve validity and agreement. The following algorithm terminates after one round. However, it picks values randomly, so it is unlikely that all of the nodes agree on the value. If input values are all 1 or all 0 output will not satisfy the validity requirement.

Algorithm 3 Anti-Paxos

1: $v_i \in \{0, 1\}$ ◁ input bit
2: Choose v_i randomly, with $Pr[v_i = 0] = Pr[v_i = 1] = 1/2$
3: return v_i

b) $C1$ stores command a on $S1$. $C2$ then stores command b on $S2$ and $S3$ and slows down. Now client $C3$ starts and supports command a, which has the smallest T_{store}, it gets stored on $S2$, while the message to $S3$ gets delayed. $C2$ can now execute command b, while $C3$ can execute a as both had achieved a majority. Command a gets executed on $S1$ and $S2$ and command b gets executed on $S3$.

c) The Changed Paxos would work if line 1 would be $t = t - 1$ and lines 3 and 9 would use $<$ comparison instead of $>$. This would result in an equivalent algorithm to the original Paxos just with tickets going from 0 to $-\infty$ instead of going from 0 to $+\infty$.

3 Agreement (20 points)

A group of friends wishes to decide whether they should hold a party to celebrate their successful exams. Since they suspect that some friends might actively want to prevent them from reaching agreement, Alice suggests an improvement to the Ben-Or algorithm, such that it works for $f < n/5$. Your task is to help the group of friends, by completing the missing value on line 21 and proving the algorithm's correctness, as depicted below.

Algorithm 4 Asynchronous Byzantine Agreement (Ben-Or)

```
1:  x_u ∈ {0,1}                    ◁ input value of node u
2:  round = 1                      ◁ round
3:  while true do
4:    Phase 1
5:    Broadcast myValue(x_u, round)
6:    Wait until n − f myValue messages of current round arrived
7:    if more than (n+f)/2 myValue messages contain the same value x then
8:      x_u = x
9:    else
10:     x_u = ⊥
11:   end if
12:
13:   Phase 2
14:   Broadcast propose(x_u,round)
15:   Wait until n − f propose messages of current round arrived
16:
17:   if at least 3f + 1 propose messages contain same value x with x ≠ ⊥ then
18:     Broadcast myValue(x,round + 1)
19:     Broadcast propose(x,round + 1)
20:     Decide for x and terminate
21:   else if at least ... propose messages contain same value x with x ≠ ⊥ then
22:     x_u = x
23:   else
24:     choose x_u randomly, with Pr[x_u = 0] = Pr[x_u = 1] = 1/2
25:   end if
26:   round = round + 1
27: end while
```

a) [4] Prove that if after *Phase 1* two correct nodes u and v have an input value different from $\perp$ (i.e., $x_u \neq \perp$ and $x_v \neq \perp$), then $x_u = x_v$.

b) [4] Find the minimal value to complete line 21 such that the following statement holds true: *If a correct node u sets $x_u = x$ on line 22, no other correct node v sets x_v to a value different from x.* Then prove that this statement holds true with your suggested value.

c) [8] Prove that this algorithm satisfies agreement.

d) [4] Bob proposes to use FIFO broadcast in line 14, instead of the regular Best-Effort broadcast. Can the amount of proposals in the condition of line 17 and line 21 be lowered while still satisfying byzantine agreement? If so, by how much?

a) Assume $x_u \neq x_v$, with $x_u \neq \perp$ and $x_v \neq \perp$. Both u and v have each seen a byzantine quorum of more than $\frac{n+f}{2} - f = \frac{n-f}{2}$ correct nodes broadcasting x_u or x_v respectively. Since the two sets must be disjoint that implies that there are more than $n - f$ correct nodes in total, which is a contradiction.

b) A = f+1. Proof: A node executing line 22 with value x, must have received at least 1 proposal for $x \neq \perp$ from a correct node. By the previous statement, no correct nodes will propose a value different from x, hence no value other than x will be accepted.

c) Let u be the first node deciding on x. By the first statement of the exercise, no correct node can decide on a different value in the same round. Next, due to scheduling, another correct node can have received proposals from a subset differing by f values. Additionally at most f proposals might come from byzantine nodes and be different from x. Thus at least $3f + 1 - f - f = f + 1$ proposals are received by all correct nodes. Using the second statement from above, we also know that no correct process adopts another value, hence in the next round every correct node will broadcast x as their input value. As $n - 2f > \frac{n+f}{2}$ every node will set $x_u = x$ and every node will also see $n - 2f > 3f + 1$ proposals for x. Thus every correct node will decide x as well and agreement is satisfied.

d) The value of line 17 can be lowered by f, to 2f + 1, as now Byzantine nodes can no longer disseminate two different messages (as in the exercise session). The value of line 21 on the other hand can't be changed.

4 Bitcoin (18 points)

Bitcoin provides a mechanism called *hashlocks* to unlock bitcoins by revealing a secret preimage to a hash string. The recipient creates X and $Y = \texttt{SHA256}(X)$, and gives Y to the sender. The sender locks their bitcoins with $\texttt{UTXO}(Y, P_R)$. The recipient gets these bitcoins with a transaction containing string X such that $\texttt{SHA256}(X) = Y$, and also a digital signature with the secret key S_R that corresponds to the recipient public key P_R.

a) [5] Why is the digital signature with the secret key S_R necessary when unlocking the coins? In other words, why can we not drop P_R from $\texttt{UTXO}(Y, P_R)$, and just unlock with the secret X?

Litecoin, another blockchain based cryptocurrency, also offers hashlocks. Alice and Bob want to exchange litecoins with bitcoins without using a trusted third party. Alice locks her bitcoins with $\texttt{UTXO}(Y, P_B)$, where $Y = \texttt{SHA256}(X)$ (only Alice knows the secret X) and P_B is Bob's public key. Bob locks his litecoins with $\texttt{UTXO}(Y, P_A)$, using the same hash string Y (Bob does not know X *at this time*) and Alice's public key P_A. Alice redeems these litecoins with a transaction containing X (which she knows from the beginning) and her own signature corresponding to the public key P_A.

b) [4] Bob can redeem his bitcoins only after Alice has redeemed her litecoins, and not before. Why?

c) [3] Security is defined as: if a party pays P coins for Q, they either get Q or get back P. Due to no fault of their own, a party should not lose P and not get Q. Why is the above scheme not secure if one party aborts?

d) [6] Can we make this scheme secure under aborts by using timelocks (discussed in the script) and a logical `OR` condition?

a) Bitcoin transactions spend an indeterminate amount of time in the so called "mempool", when they are moving around from peer to peer in the Bitcoin network before being included in the next block by a miner. The recipient's transaction that claims the $\texttt{UTXO}(Y, P_R)$ by revealing X in plain text is visible to all nodes in the network. If the recipient's transaction is not encumbered by a signature as well, anyone else can craft their own transaction that also reveals this now public string X and try to claim the bitcoins for themselves. This creates a race condition between the two (or more) such transactions and there is no guarantee that the original recipient gets the bitcoins in the end. Digital signatures make sure that only the intended recipient can sign the receiving transaction with their knowledge of the secret key S_R. Note that S_R is never broadcast on the network, unlike X, which is a part of the transaction.

b) Bob doesn't know the secret preimage X till Alice's redeeming transaction is confirmed on the blockchain. After it is confirmed on the blockchain, which is public data, Bob can inspect the blockchain and learn X, which he can use to construct his redeeming transaction.

c) If Alice aborts before redeeming Bob's litecoins, Alice loses her own bitcoins, but due to no fault of his own, Bob cannot get either Alice's bitcoins or his own litecoins back. His litecoins are locked by Y, and Bob doesn't know X.

d) Alice and Bob both construct their $\texttt{UTXO}'s$ with an `OR` condition. The `OR` condition has a timelock "arm" on one side and a hashlock "arm" on the other side. The timelock "arm" of the transaction sends their own money back to them after time T, and the hashlock "arm" of the transaction sends the money to their counterparty. If Alice doesn't redeem Bob's litecoins using her knowledge of the secret preimage X within time T, Bob can use the timelock arm and refund his coins back to himself.

5 Game Theory (16 points)

Alice and Carol went on a trip and bought two identical antique vases for 500 Francs each. Unfortunately, their luggage was not handled properly when flying back home, and both vases broke. The airport manager wants to compensate Alice and Carol for the broken vases. As he does not know the true price of such a vase, he proposes a scheme.
Alice and Carol are placed in separate rooms so that they cannot communicate. Each of them is asked to write down the price of one vase as a natural number between 20 and 1000. Then, let p_{Alice} and p_{Carol} denote the prices they wrote and let Δ denote an integer ($0 \leq \Delta \leq 20$). They will be compensated as follows:

- If $p_{Alice} = p_{Carol} = p$, then p must be the true price of the vase and Alice and Carol receive p Francs each.
- If $p_{Alice} > p_{Carol}$, the airport manager assumes that Alice lied and Carol's price is correct. Then, Alice receives $p_{Carol} - \Delta$ Francs, and Carol receives $p_{Carol} + \Delta$ Francs.
- Symmetrically, if $p_{Alice} < p_{Carol}$, the airport manager assumes that Carol lied and Alice's price is correct. Then, Alice receives $p_{Alice} + \Delta$ Francs, and Carol receives $p_{Alice} - \Delta$ Francs.

Alice and Carol are both smart and rational. Each of them wants to maximize their own compensation, even if it means to lie to the airport manager.

a) [3] Is writing down the true price of the vase a dominant strategy for Alice when $\Delta = 5$? Why?

b) [3] Find a pure Nash Equilibrium for $\Delta = 5$.

c) [5] What is the Optimistic Price of Anarchy ($OPoA$) for $\Delta = 1$?

d) [5] For which values of Δ is there a unique pure Nash equilibrium? Justify your answer.

a) No. A strategy is dominant for Alice if she is never worse off by playing this strategy. If $p_{Carol} = 499$ and Alice writes down the true price of the vase ($p_{Alice} = 500$), then Alice only receives 494 Francs. However, if Alice writes down 498 instead, she receives 503 Francs.

b) For $\Delta = 5$, there is only one pure Nash Equilibrium: $p_{Alice} = p_{Carol} = 20$.

c) Recall that the Optimistic Price of Anarchy is defined as the ratio between the highest payoff among the Nash Equilibria and the payoff of the social optimum.

We first compute the payoff for the social optimum: The total payoff of any strategy profile is $2 \cdot \min(p_{Alice}, p_{Carol}) \leq 2 \cdot 1000$. Then, a social optimum is achieved when both Alice and Carol write down 1000, and the total payoff is 2000.

The highest payoff among the Nash Equilibria is at most 2000, and in fact equal to 2000. This is because $p_{Alice} = p_{Carol} = 1000$ is a Nash Equilibrium: one player writing down a different price leads to the same or a lower compensation.

Therefore, the Optimistic Price of Anarchy is $\frac{2000}{2000} = 1$.

d) The game only has a unique pure Nash Equilibrium when $2 \leq \Delta \leq 20$: $p_{Alice} = p_{Carol} = 20$.

When $2 \leq \Delta \leq 20$ and $p_{Alice} = p_{Carol} = 20$, none of the players has the incentive to unilaterally change their choice. Writing down a different number results in a compensation of $20 - r$ Francs instead of 20. Hence, this strategy profile is a pure Nash Equilibrium.

We now show that any other strategy profile is not a pure Nash Equilibrium when $2 \leq \Delta \leq 20$. If $p_{Alice} > p_{Carol} \geq 20$, Alice's compensation is $p_{Carol} - r$ Francs. Alice has the incentive to change her choice to $\max(p_{Carol} - 1, 20)$ and then receive at least p_{Carol} Francs. The same argument applies for Carol when $p_{Carol} > p_{Alice} \geq 20$. If $p_{Alice} = p_{Carol} = p > 20$, Alice and Carol have the incentive to unilaterally change their choice to $p - 1$ and receive $p - 1 + r$ Francs instead of p.

If $\Delta = 1$, the game has multiple Nash Equilibria. For example: $p_{Alice} = p_{Carol} = 1000$ and $p_{Alice} = p_{Carol} = 500$. One player writing down a different price leads to the same or a lower compensation.

Similarly, if $\Delta = 0$, the game has multiple Nash Equilibria, such as $p_{Alice} = p_{Carol} = 1000$, or $p_{Alice} = p_{Carol} = 500$. One player writing down a different price leads to the same or a lower compensation.