modified	subject	message
Monday, 2 November 2020, 6:12 PM	Re: Recording for part II	You can find the new recording in the "Lecture recording" link provided below Week 8/Monday.
Monday, 2 November 2020, 5:24 PM	Recording for part II	Dear all, is the new zoom link (for part II, starting with today's lecture) again https://video.ethz.ch/lectures/d-itet/2020/autumn/252-0217-00L.html? There was one video uploaded today, but it is two hours of black screen with someone cleaning the blackboard in the end.
Friday, 6 November 2020, 3:49 PM	Re: How does client-side ticket number work?	oh, I see. Exactly, Paxos addressed it. Thanks
Friday, 6 November 2020, 2:31 PM	Re: How does client-side ticket number work?	If you look at Paxos (Algorithm 15.13) it actually addresses the problem of uneven server tickets using client-side tickets. The important thing is that the client proposes the same ticket to all of the servers. It then proceeds only if the majority of the servers issue that ticket. If the client doesn't receive the tickets from the majority it tries again with a larger proposed ticket. If the majority issues that ticket it means that at that point the ticket numbers of majority of the servers are "~synced up". Because the process is asynchronous until you receive all the tickets some of them might get overwritten but hopefully you can see how this forces the majority of the servers to have a similar largest ticket number. For client to store a command it has to receive a ticket from the majority. So if there is a command stored with currently largest ticket number, when it was stored, majority of the servers must have issued that largest ticket -> command is newer than any other.
Friday, 6 November 2020, 8:58 AM	How does client-side ticket number work?	In chapter 15 on page 142, I don't quite get the last remark. I can understand that each server may use its own ticket numbers, e.g. server 1 issues ticket starting with 10001, and server 2 ticket starting with 50001. But i don't quite understand why client-side ticket can solve this?
Friday, 6 November 2020, 3:50 PM	Re: Why does hard link not require disk space?	It makes sense now. Thanks.

Friday, 6 November 2020, 1:26 PM	Re: Why does hard link not require disk space?	Yes the directory entry consumes some disk space, but we attribute this to "the space the directory uses" and not to "the space the file uses".
Friday, 6 November 2020, 9:00 AM	Why does hard link not require disk space?	Isn't it stored in the inode of this file's directory (which is also a file itself storing in disk)?
Friday, 6 November 2020, 1:25 PM	Re: How can symbolic link point to file on other machine?	Symbolic links really just contain another (possibly relative) name, that is then resolved again. The binding of that name might change when you perform mounts. A symbolic link is resolved starting from the location of the symbolic link. Hence it cannot point to a file on another machine, unless you have mounted the remote machine somewhere in your FS namespace. So let's say you mount the remote machine as "/remote" then you can distinguish between "/etc/passwd" for the local file and "/remote/etc/passwd" as remote file. You can mount some remote /etc over you local /etc, but then you lose the ability to refer to your former local /etc/passwd, both in the file system and in symbolic links.
Friday, 6 November 2020, 9:05 AM	How can symbolic link point to file on other machine?	To be more specific, how does the OS know it doesn't point to a file on this machine which could have the same name due to using same name space? e.g. /etc/passwd
Friday, 6 November 2020, 3:44 PM	Exercise Session Slides	Dear all, As the exercise session slides involve many references / questions to code written on previous slides, would it be possible to upload the session slides at the beginning of the exercise session? This would be very great. Thank you, Tassilo

Friday, 6 November 2020, 5:51 PM	Re: Improving performance by adding negative replies in Paxos	Yes, the main benefit is that you no longer have to wait for the whole timeout period. Which in a real implementation would be much larger than the mean message delivery time. So over many messages we potentially would see large decrease in time till convergence. The negative reply can also carry extra information, such as current maximum accepted ticket number. Which could allow the client to save a few rounds. As there is no point in sending ticket requests if the proposed ticket number is smaller than the server's current maximum. All of this is more of a practical consideration - one of the ways you would improve the algorithm if you wanted to implement it. It's true that under theoretical asynchronous setting sending a negative reply with no extra information is basically the same as not sending one as in both cases the client would have to potentially wait 2 time units (time to and from server) until it knows that its ticket request or proposal was rejected. However, if the negative reply also tells the client current highest ticket number there is a potential to save some rounds.
Friday, 6 November 2020, 4:44 PM	Improving performance by adding negative replies in Paxos	On script page 144, the second last remark about Paxos says: >>The performance can be improved by letting the servers send negative replies in phases 1 and 2 if the ticket expired. I don't see how the improvement is achieved exactly. My thought so far is that: given negative replies may help the client differentiate "message loss or slow message" from "not the freshest ticket". As a result, it can start generating new tickets once receiving the negative reply(i.e. restart with phase 1). The only potential improvement I can see here is that the client doesn't need to wait the whole time unit (in asynchronous setting) to finally realize it has to restart. Is this the only benefit we may have by adding negative response? If so, I think the benefit is limited since the negative reply could also delay in our setting. Could you comment on this
Saturday, 14 November 2020, 2:06 PM	Re: Why does 0-valent's child have to be 0-valent?	Yes, this is correct.

Friday, 13 November 2020, 7:10 PM	Re: Why does 0-valent's child have to be 0-valent?	Thanks for your answer. Now what I understand is: by definition, 1-valance means the algorithm must converge to 1; And the transaction from configuration to configuration is simply the process of running this algorithm round by round. Therefore, all descendants of a 1-valance must be 1-valance. I think I have messed up some definitions in the previous reasoning, hopefully I got it correctly now. If this is not the case, please point it out. Thanks a lot for your help ;)
Monday, 9 November 2020, 11:30 AM	Re: Why does 0-valent's child have to be 0-valent?	If we assume no node failures and the algorithm knows it your reasoning would be correct. However, do note that a consensus algorithm doesn't even have to choose the majority value. For example if we make Algorithm 16.15 deterministic, by always setting v_i = 1 in line 19 we can end up with a consensus of '1' even though initially we had a majority of '0'. What 0-valance or 1-valance means is that for the current algorithm it's impossible to converge to any other value. For example, if we again consider the modified deterministic version of Algorithm 16.15 and have a majority of '1' there is no way for it to converge to '0' anymore. If a configuration is 0-valent all of its children will also be 0-valent, same for 1-valance.
Friday, 6 November 2020, 6:13 PM	Why does 0-valent's child have to be 0-valent?	In proof of Lemma 16.13 on script page 151, it mentions: >>The child configuration that follows a 0-valent configuration must be 0-valent. I would like to check if I got the reasoning correctly. 1. Given a 0-valent configuration C, it indicates that the majority of nodes has value 0, e.g. V=[0, 0, 0, 1, 1] if assuming n=5. 2. In the child configuration of C (denoted by C_child), since node with value 0 will only send 0, the majority will remain 0 (i.e. the number of node with value 0 cannot decrease). 3. Hence, C_child will stick to 0-valent. 4. In addition, it also indicates that after getting into a univalent configuration, any descendant of it will remain univalent with the same value. Is this correct?

Monday, 9 November 2020, 10:46 AM	Re: Always choosing 1 prevents from termination	It is true that in Lemma 16.19 we only have expected number of iterations and no hard guarantee on number of execution steps (we could provide a number of iterations needed to terminate with an arbitrarily high probability using i.e. Markov's inequality). The fundamental difference is that if the algorithm is given a worst-case sequence of events (say by an attacker) the deterministic algorithm will NEVER terminate while the randomized algorithm always has a chance to 'escape' a bad scenario using randomness and terminate. In algorithm analysis it's common to focus on the worst case scenario to prove algorithm correctness/termination. The randomised algorithm can terminate with any sequence of events, while the deterministic has some sequences under which it does not terminate.
Friday, 6 November 2020, 7:42 PM	Always choosing 1 prevents from termination	It is mentioned both in the lecture and in the script that making the line 19 deterministic of algo16.15 doesn't work because termination cannot be guaranteed. (*) However, what confuses me is that in Lemma 16.19, we don't have a fixed termination time ,either (it just shows the randomized algorithm will terminate with high probability). So even with the randomized algorithm, there is no hard guarantee on termination time. May I understand * in this way: by always choosing 1 or 0, the expectation of termination time is even longer (or say: the algorithm will terminate with lower probability within the same limit of rounds)?
Tuesday, 10 November 2020, 1:57 PM	Re: Recordings of Friday lecture	Does this apply to recordings of the exercise session as well?
Monday, 9 November 2020, 7:07 PM	Re: Recordings of Friday lecture	Just to let you know, so you do not ask every time: The recordings from the lectures on Friday will be uploaded the earliest the following Monday, and the latest by the end of the following week due to the processing by the administration.
Monday, 9 November 2020, 6:27 PM	Re: Recordings of Friday lecture	Finally, it's uploaded.
Monday, 9 November 2020, 5:00 PM	Re: Recordings of Friday lecture	still not?

Sunday, 8 November 2020, 1:21 PM	Re: Recordings of Friday lecture	Hi Gregor, The recording is not ready to upload yet. When it is ready, there will be a link in moodle like the one from Monday's lecture. Best Zeta
Sunday, 8 November 2020, 12:36 PM	Recordings of Friday lecture	Dear Computer Systems team, I wanted to ask where I can find the recording of the latest lecture. It seems like the monday one is located on moodle, but I cannot yet find the friday one. Best, Gregor
Thursday, 12 November 2020, 10:24 PM	Re: Time units vs. no upper bound on slowest message	In essence, we assume that messages take up to 1 time unit, but the nodes don't know what that time unit is. So from the node's perspective it doesn't know if the message is just taking much longer than other messages did or if the sender node failed. However as you noted we do need this definition that messages take at most 1 time unit to be able to do any analysis on time complexity for example. But it's only for the analysis and not to be used in the algorithm itself. Note that knowing real maximum possible message delay in the real world is also impossible, we can only know the historic average and the deviations from it, but that will never encompass all of the future messages with absolute certainty.

Thursday, 12 November 2020, 9:21 PM	Time units vs. no upper bound on slowest message	This is a general question: We seem to use a mixture of two different concepts of time units / unbounded delays. The proof that there is no f>= n/2 consensus algorithm for the asynchronous model (Thm 16.21) relies on the assumption that messages could take very long between two partitions. At some point in time, we must stop waiting, as the other nodes simply could have failed. However, time units were defined previously by the longest time period a message takes. Every message is sent within one time unit. Many previous proofs rely on this definition, for example when determining the runtime. So how does this definition of time units harmonize with the problem of not knowing if we are still waiting for a slow message of that node failed? The usage of time units assumes an upper bound on _any_ message delay, while Thm 16.21's proof shows that we cannot assume that there is an upper bound. Thanks in advance
Friday, 13 November 2020, 5:55 PM	Re: Inconsistency on exercise soluction 7.1.2.a, when does B send last message exactly?	If we read the algorithm as stated, node B always waits 2 seconds (independent or RTT) so the Figure 1 is correct and it should be T_0+8.5. We will correct the text, thank you for noticing.
Friday, 13 November 2020, 4:18 PM	Inconsistency on exercise soluction 7.1.2.a, when does B send last message exactly?	In the answer of a, it states: B sends the last "execute" at T_0+7.5. But in the figure below, it sends at T_0+8.5. Which one is correct? or we accept both? Both time point make sense to me. It says that all message will arrive within 0.5 sec, so the RTT is 1 sec, add it to the second last message, we get T_0+7.5; However, if we strictly follow the algorithm description, the waiting time is fixed to 2 sec for B, so T_0+8.5. Of course, one could argue that, given the majority already replied, there is no need for B to wait until the end. It seems to be a trivial question because I cannot imagine any scenario where it will cause problem (since the execution of command doesn't disturb any procedure). But I would appreciate it if you could clarify this.
Sunday, 22 November 2020, 1:11 PM	Re: Simple Synchronous Byzantine Shared Coin - Public key	Thank you, that makes sense for me!

Monday, 16 November 2020, 2:29 PM	Re: Simple Synchronous Byzantine Shared Coin - Public key	Well the dealer code, has to be run also before the algorithm starts. As you pointed out, there is no difference with an oracle if the dealer runs during the execution of the consensus algorithm. You can even use this dealer to do even better things, if such a trusted dealer would exist. Consider the following consensus algorithm. 1. Every node sends his input to the dealer. 2. The dealer receives these values, and sends the decision to everyone. 3. The nodes receive the decision, and output this value. That's it, with only three lines, we have solved consensus (no randomization, nothing). But, this is because we are assuming a trusted (central) party that we know for sure won't behave maliciously. So, the algorithm with the dealer makes sense only if it is a precomputation phase. You can imagine for example a scenario where when the "machines" are built they can have some shares for future coins which may be used for many times to solve consensus. Finally, the PKI setup is also somewhat different from the dealer (even though, in both settings we are giving some prior information to the nodes). For example, in PKI the channels can be public, since no secrets are exchanged among the parties. The one with the dealer, the channels are assumed to be private. It is all about how reasonable one assumption is and depending on the application, which one is the best for you.
Monday, 16 November 2020, 1:26 PM	Re: Simple Synchronous Byzantine Shared Coin - Public key	I understand, thank you very much! Is this different from a dealer, which we want to avoid, because the dealer has to do work during the execution of the algorithm, while exchanging public keys can be done before the algorithm even starts? Related to this, could we just replace the dealer with an oracle, and have the same effect? I don't see much difference between those two implementations.
Monday, 16 November 2020, 12:34 PM	Re: Simple Synchronous Byzantine Shared Coin - Public key	Short answer: It is not allowed to choose the key-pairs "on the fly" (since we assume a PKI setup, that is every node has a key-pair for signing, which is known to every other node). This assumption is very important. In fact, the setup preparation is the main assumption here that allows us to do something like this. If you generate random coins on the fly, as you have shown, things don't work quite as we want (even though we are using cryptographic schemes). The "setup" assumption (giving the nodes some prior information) makes the difference.

Saturday, 14 November 2020, 8:48 AM

Simple Synchronous Byzantine Shared Coin - Public key

Dear TA team,

I was wondering if the nodes were allowed to chose their public keys at the start of the round. If that is the case and I control f=n/3-1 evil nodes, wouldn't it be possible to keep generating keypairs during that phase, until I am happy with them? As an example:

Our evil nodes are called f1...fm.
We pair them up, so f1, f2 will be the first pair.

Now we want to generate keys for them, so that hash(msg(1)_f1) ends with a 0 bit, and it's smaller than 99% of all possible hash values.
hash(msg(1)_f2) ends with a 1 bit, and it's smaller than 99% of all possible hash values.
Generating both will take 200 times as long as I need if I don't care about the value, but that's okay.

Now, I have a good shot that both those nodes will have a lower hash value than all other nodes. (unless n is huge, then just make it 99.9% instead, until we do have a good shot - we can take as much time as we want in the generation process after all?).
We can now decide to only send in the value of one of those nodes, to decide the outcome of that round.

Repeat the process, but with the second pair f3,f4 and with msg(2).

This way, we can make it as unlikely that the algorithm terminates in the first m/2 rounds. And the more time we have to generate keys, the more unlikely we can make it. So in expectation, the algorithm doesn't terminate in O(1), but in O[No] instead.

Is there any mistake in this approach? Or would this work? And is there a way to prevent this from happening?

Tuesday, 17 November 2020, 8:15 PM

Re: Zoom recordings

Yes, I will try to do so.

Best,
Zeta

Tuesday, 17 November 2020, 7:40 PM	Re: Zoom recordings	I understand, thank you very much! Is it possible to do so on both days, so that even more can benefit from it? Best, Gregor
Tuesday, 17 November 2020, 12:32 PM	Dear TA team,+H38 I was wondering if the nodes were allowed to chose their public keys at the start of the round. If that is the case and I control f=n/3-1 evil nodes, wouldn't it be possible to keep generating keypairs during that phase, until I am happy with them? As an example: Our evil nodes are called f1...fm. We pair them up, so f1, f2 will be the first pair. Now we want to generate keys for them, so that hash(msg(1)_f1) ends with a 0 bit, and it's smaller than 99% of all possible hash values. hash(msg(1)_f2) ends with a 1 bit, and it's smaller than 99% of all possible hash values. Generating both will take 200 times as long as I need if I don't care about the value, but that's okay. Now, I have a good shot that both those nodes will have a lower hash	Hi Gregor, Unfortunately, we cannot let you record the session locally. But I can do the following: I can upload the link to the "raw" material on Friday after the lecture, and then replace it with the "processed" version when I receive it, so you have access to the recording during the weekend. Does this sound good? Best, Zeta

Saturday, 14 November 2020, 8:52 AM	Zoom recordings	Dear TA team, I have already asked it before, but I forgot to give the TA a way to get back to me. What I wanted to ask for, is to allow us students to record the lecture via zoom. I don't think there is a disadvantage to this, as we get the recording any way, but this way we can get it faster. As seen in https://moodle-app2.let.ethz.ch/mod/forum/discuss.php?d=62276, quite a few people were inpatient about the recordings. Letting us also record would solve this problem. (The reason why so many asked about the recordings is that some of us can't make it to the Friday lecture, but would like to join the Monday one, which only makes sense if we were able to catch up with the content already.) Best, Gregor
Monday, 16 November 2020, 12:54 PM	Re: King Algorithm- Algorithm 17.14	If I understand your question correctly, you want to know whether King Algorithm works for any input and not just binary input? The answer is: King Algorithm works for any input values (for example, node 1 has input 1, node 2 has input 2, node 3 has input 3, and so on). There can be some phases during the protocol where the parties simply don't propose anything. However, once the king is correct then all honest parties will have the same input (in the next round) and from there they will never change.

Saturday, 14 November 2020, 12:09 PM	King Algorithm- Algorithm 17.14	In case of King Algorithm do correct nodes decide just on 0 or 1 (or any other two values) or different values? In case of different values (e.g. each correct node has its own value) if each correct node has its own value then they cannot agree on one value (unless they have their own algorithm to pick up the smallest one). Is that correct that in the first round in the phase Proposal some nodes should receive value [Yes] already n-f times and then it is just a matter of guiding of the rest correct nodes to this value, it will be eventually done in the round with the correct node as the king. That is my understanding but it works only if there are two initial values. Please correct me if I am wrong here. But in case the nodes have their own algorithm like pick up the smallest one where in the code of Algo 17.14 it will work. Because my understanding is still that in the first round the some nodes should propose[Yes] in Proposal phase. Or it is not correct? Many thanks!
Monday, 16 November 2020, 1:22 PM	Re: Algorithm 17.9: Modification to achieve All-same validity	There are two different problems/mistakes related to your question. 1. First, the interpretation of "choose a value proposed by at least two nodes", has nothing to do with the choice of T. T is still the same (Line 8 is changing, which happens after T is constructed). The clicker question was about how to choose the value in T, after T is already computed. In this case, the message tuple(b, 1) would not be in T, because c_x would have to "hear" the value tuple(b,1) twice, which doesn't happen. So this value, will not be included in T at all. However, the value tuple(c_y, 1) will be in T. "choose a value proposed by at least two nodes" means that after T is constructed, we check whether some value in T was proposed by two different nodes (each of those values is heard twice to be included in T in the first place). 2. All-same validity says that if all nodes have the same input, then they keep this value as an output. In your example, the nodes don't have the same input, c_x has input 3, c_y has input 1, and c_z has input 2.

Saturday, 14 November 2020, 11:04 PM	Algorithm 17.9: Modification to achieve All-same validity	My question is regarding Algorithm 17.9, specifically the approach mentioned in a clicker question on how to achieve all-same validity. To me, it is ambiguous to what "choose a value proposed by at least two nodes" means. To me, the the following interpretation makes the most sense. We choose T as the set of all \(x.\exists v. \text{x appears at least twice in } S_v \). Based on that interpretation, I came up with the following scenario where the correct node \(c_x\) receives the value 1 twice, i.e. 1 is contained in \(S_{c_x}\) twice. Therefore, \(c_x\) would decide on value \(1\) despite it being proposed by a byzantine node. This is in violation with the correctness of that clicker question answer, I would gladly appreciate a clarification on where I'm wrong. Thanks!
Monday, 16 November 2020, 1:47 PM	Re: All-same validity: What if not all correct nodes agree on the same value?	Byzantine Agreement is defined by three properties: (All-same) Validity, Agreement and Termination. If all nodes don't have the same input, the agreement property still makes sure that at the end of the protocol all parties have the same output (no matter which value it is). So, for example if we have the binary case (where all nodes have as input either 0 or 1), because of all-same validity we know that at the end of the protocol if all nodes start with 0, they keep this value. The same way, if they start with 1, they keep 1. Finally, because of agreement, if some have 0 and some 1, then they will output consistently some value 0 or 1. As we can see, this captures (at least in binary setting) all what we want. If it is not a binary setting (for example, nodes can have input 0,1 and 2), then what all-same validity tells us is that: If some correct nodes start with 0 and some with 1, then parties in the end can still decide for 2 (which may or may not be what we want). The reason why we do this, is that having a stronger definition for all-same validity makes things harder. Say for example, every party has different inputs, how can we make sure that the value that the nodes decide, originates from a correct node? There is no way to distinguish those values from the byzantine nodes. However, as I described before, for the binary case it is already equivalent to "Correct-input

Saturday, 14 November 2020, 11:17 PM	All-same validity: What if not all correct nodes agree on the same value?	Hi! This question seems super basic, but I would nevertheless be glad to get confirmation. Namely, as the all-same validity states: \((\forall \text{correct }v. input(v) = x) \Rightarrow \forall \text{correct }w. decision(w) = x \) Above representation seems faithful to the definition given in the script and it shows that the case where not all correct nodes have the same input is left undefined. Although common sense tells my that the nodes should then regard the round as failed, this is not made explicit. Hence, my question: Can we transform the definition of all-same validity into an if-and-only-if, or is the definition on purpose phrased in such a way that that the case of disagreeing correct nodes is undefined? Thanks!
Monday, 16 November 2020, 2:08 PM	Re: King Algorithm - Distinct input values and first king byzantine	Yes you are right. But, the main question here is what "valid" really means. Even though the parties choose the value "n+1", still all-same validity is not violated. Furthermore, agreement and termination is also fulfilled. So, all is fine. However, please note that the algorithms typically assume an input set for example I = {0,....,K}, and then one value in this input set will be chosen. In your case, if I = {1, ..., n+1}, then indeed with your strategy the parties will decide for n+1, but there is nothing wrong with that according to the definition of Byzantine Agreement. The definition you are referring to is called "Correct-Input Validity", but that is too strong for our case, so we don't use that here.

Sunday, 15 November 2020, 1:28 PM	King Algorithm - Distinct input values and first king byzantine	My question, although similar to a previously asked question, tackles an issue not discussed in the lecture or on moodle. Specifically, assume that we have n nodes \(v_1, v_2, ..., v_n\), node \(v_i\) picks \(i\) as its initial value and \(f\) of these nodes are evil. Furthermore, assume that we just pick kings in ascending order, so the first king is \(v_1\), which for sake of my argument is an evil node. Given that situation, clearly lines \(4, 7\) have an unsatisfied condition and therefore the branch is not executed. As we assume that the king is bad, it can propose value any value, e.g. \(v_{n+1}\) which due to line \(13\) is then chosen by all correct nodes. Therefore, although all nodes agree on a value, that value is not valid as no node had it as an input.
Sunday, 22 November 2020, 1:10 PM	Re: Why does the dealer generate n coinflips?	Thank you very much! I took some time to look over this topic again - if I understand correctly, after n rounds, if we have really bad luck, we just continue to execute our algorithm as we would have if there never was a dealer?
Monday, 16 November 2020, 2:00 PM	Re: Why does the dealer generate n coinflips?	The shares of the coins have to be generated before the algorithm of consensus starts. The reason, why the dealer is not part of the protocol during consensus is that if that would be the case, then we can simply let the dealer choose the value to output and solve consensus in one step (since we trust the dealer to be honest). Why do we have exactly n? Well, that can be also changed, but the more coins are preprocessed the better.
Monday, 16 November 2020, 1:30 PM	Why does the dealer generate n coinflips?	I understand that the dealer has to generate a coin at some point during every round, so that nodes can have their "random" value. But in algorithm 18.23 we see that we generate n such coins. Why is that the case, and what are they good for? Thank you for your help!

Tuesday, 24 November 2020, 3:32 PM	Re: Proof of theorem 18.28	Yes you are right in both questions. We will fix the lecture notes. Regarding the first question: The idea is that there also exist protocols for byzantine agreement that are better than f < n/10 (not shown in the lecture) and then we could plug the same shared coin into that algorithm as well and get better bounds. In our case, we keep the worst bound (f < n/10). So, as you correctly said, strictly speaking, in our case we actually have a better bound and say with probability 9/10 the minimum hash belongs to a correct node, which would give a better bound. Regarding the second question: Yes the sum is not written in the best way and is not intuitive. The way I interpret this, it computes the average length by adding to the sum 1 + 2 + 4/3 + 8/9..., by simply saying I can have longer rounds if I wasn't lucky so far which happens with probability (2/3)^i. This is not exactly the same as the traditional formula how to compute the expected value of a geometric distribution, but leads to the same result. In the lecture notes, the result is 5 (which
Tuesday, 17 November 2020, 12:48 PM	Proof of theorem 18.28	My questions are regarding the proof of theorem 18.28: First, I am confused by the fact that in the theorem it states that we can tolerate up to f < n/10 byzantine failures, but in the proof we say that with probability 1/3 the minimum hash value is generated by a byzantine node. Would according to what we are trying to proof 1/10 not be a better upper bound for this probability, or is there another reason why we can only use 1/3 as an upper bound instead of 1/10? Secondly, still assuming a success probability 1/3 as given in the proof (i.e. the probability that consensus is reached in the next round), I don’t see why the number of expected rounds is given by the stated expression (1 + \sum_{i=0}^\infty 2 \cdot (2/3)^i = 5). My reasoning would be the following: Let X be the random variable that denotes the number of shared-coin-executions needed until consensus is reached in the next round. Then X has a geometric distribution with probability 1/3, which implies an expected value of 3. The total number of rounds would be given by 1 + 2X (since each invocation of the shared coin uses 1 round and the rest of the algorithm uses another round). By linearity of expectation, we would end up with an expected value of 1 + 2 \cdot 3 = 7 rounds instead of 5. Thanks!
Monday, 7 December 2020, 11:11 AM	Re: Understand two checks in Algo. Node Receives Transaction	Yes you are correct. If a transaction is in the blockchain (the longest) and its outputs are unspent (that is, it is a UTXO) they can be spend later as an input for another transaction. The transaction is executed locally but also broadcast so that others can execute it as well.

Friday, 27 November 2020, 2:12 PM	Understand two checks in Algo. Node Receives Transaction	On page 230 of script, there are two "if... then..." checks in the algorithm 22.13. I would like to know whether I understood it correctly. The first round of check makes sure that all input of this transaction refers to some output that this node knows (has knowledge/information locally). The second round focuses on whether the value that this transaction is going to spent is smaller than the unspent value (it is like we check we have enough money before buying a coffee). After passing these two round checks, the transaction will be executed locally. Do I understand this correctly? I am not sure because I don't know whether I get the correct meaning of "value of input" and "value of output", which, to my current understanding, both refer to the total amount of unspent outputs in some ways.
Sunday, 29 November 2020, 10:24 AM	Re: Lemma 17.16.	1.) I have found the definition of the King Algorithm in [1], which I present below. They have slightly different terminology, where the rounds 1-3 correspond to Vote, Propose, King in Algorithm 17.14. Nevertheless, it seems to be the exact same algorithm with one exception: Algorithm 17.14. states in line 12: "if received strictly less than n-f propose(y) then" where the analogous part in the slide states "if own value received < n-f proposals". So, either Algorithm 17.14 or the slides have a typo, I guess. If Algorithm 17.14 has a typo, I think, it should state "if received strictly less than n-f propose(x) then". 2.) Now, consider the example run of the King Algorithm with n=4, f=1 below Suppose now, that in phase 1, round 3 the byzantine king node would send 0 to every node (since it's byzantine, this is allowed). Then in phase 2, round 2 it would happen that all correct nodes would propose 0. This then would contradict lemma 17.16., because in phase 1 some correct node has proposed value 1. Therefore, I think lemma 17.16. should state: FOR EVERY PHASE, IT HOLDS THAT: If a correct node proposes x, no other correct node proposes y, with y /= x, if n > 3f. [1] http://www.csc.kth.se/utbildning/kth/kurser/DD2451/pardis11/DD2451_lecture7.pdf

Saturday, 28 November 2020, 11:09 PM	Lemma 17.16.	Hi there, Lemma 17.16 has to do with algorithm 17.14. (The King Algorithm). It states: If a correct node proposes x, no other correct node proposes y, with y≠ x, if n > 3f. Are we considering only a single round here? Meaning that the lemma should state: For every round, it holds that: IF A CORRECT NODE PROPOSES X, NO OTHER CORRECT NODE PROPOSES Y, WITH Y≠ X, IF N > 3F. Is it possible that in two different rounds, two different values can be proposed? Kind regards
Friday, 4 December 2020, 5:48 PM	Re: Can NE always be determined by the node itself?	This line you reference is talking about the initial road network that does not have the ultra-fast 0 cost connection between nodes u and v. In that case 500/500 is the Nash equilibrium because if I choose the path that has 501 cars on it it would be better for me to move to the one that has 499 cars as the total cost to me would be lower. So while other distributions of cars are possible they are not Nash equilibrium as each driver could improve it's own cost by picking the less congested path. So when there is no ultra-fast connection between u and v Nash equilibrium is also the social optimum. But when we introduce that ultra-fast connection it's no longer the case. Yes Nash equilibrium is a particular strategy profile/game state. With a property that no player could have done better given that other players would not change their action.
Friday, 4 December 2020,	Re: Can NE always be determined by the node itself?	PS. I think my doubt is more about the question without building the "cheap fast way", so no connection between the two middle nodes u and v.
Friday, 4 December 2020, 4:26 PM	Re: Can NE always be determined by the node itself?	Thanks for the reply. The "game" part really helps me understand NE concept. However, I am still confused by the Braess' Paradox. As you said, the 500/500 situation is actually the social optimum. I agree with you on this from the definition of social optimum. What confused me was that, on the script (page 244, the third last line), it says "In the Nash Equilibrium, 500 drivers first..." So it seems that the script treats this as NE (while it is also social optimum in this specific car example). Therefore, I had my previous doubt. From your answer, my current understanding is that NE is a concept of system state, not about the individual decision. Please correct me if I got it wrong.

Friday, 4 December 2020, 3:49 PM	Re: Can NE always be determined by the node itself?	You can understand Nash equilibrium (NE) in a game that has one round as everyone picks an action at the same time, then everybody reveals their actions and the players can see what reward they get. Then if no individual player could change their action and get a better reward if other player choices stay fixed we are in the Nash equilibrium. So NE is a state that we reach if all the rational players do what's good for them individually. This is what happens in Braess' Paradox. All of the cars know that one road segment costs one while the alternative costs at most 1. So it's natural for the car to greedily pick the one with the better expected reward. What you say in the last paragraph is actually the social optimum - when half the cars go trough the top path and half through the bottom one and no one takes the ultra-fast connection (of cost 0). To stay in this social optimum cars would have to cooperate with each other. As for any individual car it's actually beneficial to use this ultra-fast connection if the rest of the cars are acting according to the social optimum. As that "cheating" car would reach the destination in 1/2 + 0 + 1/2 instead of 1/2 + 1 cost. So these greedy actors would then again push the system towards the Nash
Friday, 4 December 2020, 9:30 AM	Can NE always be determined by the node itself?	From the definition of NE in the script, it gives me a feeling that what a node should do in NE scenario can be determined solely by the node itself (without knowledge of other nodes' decision). But it seems that this is a misunderstanding. In the lecture, I thought the prof. quickly mentioned that, in some scenarios, "the node needs to be told what to do". In addition, in Braess' Paradox, I think it would be impossible for the car to know which road it should choose if other cars' choices are not known to it. Therefore, my understanding so far is that sometimes NE can only known for the whole system (e.g. 500 should go up, 500 should go down), but not the choice of a specific node (e.g. cannot say for sure whether a specific car should go up or down because it doesn't know others' situation). Is this correct?
Thursday, 10 December 2020, 11:34 AM	Re: How evil can Byzantine node be? - question on solution 9 problem 2.2	I think the misunderstanding comes from this: The agreement condition in the exercise says the "interval size" converges to zero, and not the "value" converges to zero. It is ok to converge to some other value that is not zero, as we have seen in previous exercises. The goal now is to reach a value where the parties agree (which is somewhere in the interval, but not necessarily 0).

Friday, 4 December 2020, 4:41 PM	How evil can Byzantine node be? - question on solution 9 problem 2.2	From the definition of Byzantine, we assume a Byzantine node can do "whatever it wants". Putting this into the context of Assignment 8 Problem 2.2.b), I don't see why the solution assume that the Byzantine node will send value smaller than -3 to node -3, -2, -1; send value larger than 3 to node 1, 2, 3... I thought it would send different value to different correct nodes in order to "disturb" the convergence. E.g. sends -1 to node -3, sends 2 to node 1... And for each round, it sends different values even to the same correct node. This was the worst case I could imagine because this behavior pattern might lead to a situation where the sum of all correct nodes can never be 0, no matter how many rounds we play. Do I misunderstand something here? Or the solution just presents a (less-evil) possible result?
Tuesday, 8 December 2020, 11:28 AM	Assignment 11 - question 1.5 PoA Classes	Dear TA, please could you explain the solution, maybe with drawings, for the assignment 11 question 1.5 in the next exercise session on Friday? Some statements made in the solution are not very clear e.g. about the same Nash equilibrium, where the formulas for cost (x) and cost (y) with hat are coming from, why x is not social optimum etc. Thank you

Saturday, 12 December 2020, 3:17 PM	Assignment 12: 1.3) b)	In the presented master-solution, it is stated that if the set A in a correct node is updated for the first time with a value x in a round $i < f + 1$, then in the subsequent round all correct nodes will be aware of the updated version of A. So far so good. But then the master solution boldy states that there cannot be the case that A is updated by some value y in the round $i = f + 1$. May I ask why ? The proof of theorem 26.3 is not applicable in my opinion: Just because $\|S\|>= f + 1$ and there is a message from a correct node n in S (and thats already a stretch in my opinion, but thats a separate thing), this by for does not trivially imply that one correct node n has decided on anything. At least not without some more in-depth formal reasoning. In fact I was thinking of the following case: Assume p is byzantine, but all byzantine nodes are behaving correct up until round f. We are now at the beginning of round f. A={x} is for all nodes the same and contains one value. Now p sends, in round f, the message value[Yes]_p to a single correct node n_0. This message will arrive at n_0's doorstep in round f+1. Now, V_x in node n_0 will certainly satisfy $\|V\_x\| >= f + 1 = i$, because in total the lines 9-10 have been executed around n times until now (in total over all nodes), so $\|V\_x\|$ should be somewhere around 3*f. Thus in round f+1, n_0 will enter the code section 9-10 and add y to A, thus updating $\|A\|=2$ and thus breaking the protocol. So did I forget to take something into account or is the master solution broken ?

Sunday, 13 December 2020, 1:49 AM

Question about last exam

Hello,

I try to solve this exercice of the exam of last year, however I am really stuck with the two first question namely the questions with udp and tcp. Has someone maybe a hint to answer this questions ?

Best regards

François Costa

This is the question

Suppose you are working on the implementation of a Unix-like operating system for a machine with

a network adaptor using conventional I/O descriptor rings. The network stack runs inside the kernel and uses BSD-style

mbuf structures to buffer packets.

The mbuf structure looks like this:

struct mbuf {

struct mbuf *m_next; /* next mbuf in chain */

struct mbuf *m_nextpkt; /* next packet in list */

char *m_data; /* location of data */