CSC2125: Homework #7

Due on November 12

Fan Long

Thomas Hollis

Problem 1

Suppose a blockchain using the Bitcoin protocol runs at a very fast block generation rate, so that only 40% of the generated blocks are on the main chain on average when the blockchain runs normally without attackers. Now there is an attacker controlling 33% of the total network hash rate. Is it possible for this attacker to deterministically launch double-spending attacks? Why?

Solution

Yes, this is possible in Bitcoin. This is because Satoshi Nakamoto's Bitcoin protocol assumes an exponential drop off in probability of a double spending attack. This exponential assumption is why we can use a threshold number of confirmations to assume that the transaction has safely gone through. However, this only holds true when the block rate remains around 10 min. Indeed, a faster block rate would cause a lower percentage of blocks on the main chain.

In this particular case, if only 40 percent of the generated blocks are on the main chain this means 60 percent of generated blocks get orphaned or abandoned. We also know that the attacker controls around 33 percent of the network hash rate. This means the attacker has half as much power as the honest nodes, which control around 66 percent. In Bitcoin, we know that the protocol desires one block every 10 min. Thus, by following the notation used in the original whitepaper, we can pose the following equations:

$$\frac{\beta}{\lambda_h} = 0.4$$

$$q \approx 0.5$$

$$\lambda_h = \lambda = \frac{1}{600}$$

$$\therefore \beta = 0.000\bar{6}$$

Since we know from section 4 of the whitepaper that the security threshold for a deterministic double spending attack is:

$$q < \frac{\beta}{\lambda_h}$$

Therefore, since in our case $q > \frac{\beta}{\lambda_h}$, we can state that this attacker can deterministically launch double-spending attacks in Bitcoin under those conditions.

Problem 2

For the previous question, if the chain uses GHOST protocol instead. All the rest conditions stay the same. Is it possible for the attacker to deterministically launch double-spending attacks? Why?

Solution

No, this is not possible in GHOST. Indeed, GHOST was designed to avoid the double-spending issues that arise when decreasing desired block time. GHOST achieves this by selecting, at each fork in the chain, the heaviest subtree rooted at the fork.

From the previous conditions, we have (as discussed earlier):

$$\frac{\beta}{\lambda_h} = 0.4$$

$$q = 0.5$$

$$\lambda_h = \lambda = \frac{1}{600}$$

However, we know from section 5 of the whitepaper that the GHOST protocol maintains the security threshold for successful 50% attacks at 1 rather than at $\frac{\beta}{\lambda_h}$ as seen before. This is expressed as follows:

$$q < 1$$

Here we can see that indeed $q < 1$ since $q = 0.5$. Therefore, the aforementioned situation would not allow attackers to deterministically launch a double-spending attack. In addition, it is interesting to note here that this security threshold is only broken when q is greater than or equal to 1, i.e. when the hash rate of the attacker is more than 50% of the total hash rate. At this point the protocol would break apart anyway so this new threshold is a strong one.

Problem 3

One claims that by using GHOST protocol, we can now simply lower the difficulty to speed up the block generation to achieve arbitrarily high transaction throughput, because fast block generation no longer harms the security. Is this claim true or not? Why?

Solution

No, this claim is not true. Indeed, section 1 of the whitepaper explains this limitation in detail.

In simple terms, the GHOST algorithm increases the difficulty of successfully implementing a double-spending attack at higher transaction rate but it does not address two other critical issues that arise with fast transaction rates.

Firstly, when block generation rate is raised too high, emphasis becomes placed on the network connection quality of miners. Therefore, as detailed in the whitepaper, better connected miners would benefit from slightly larger rewards than their share of hashing power. This unfairness would not only disincentivise weak miners by artificially pushing toward centralisation but is also a serious security flaw. This is because 50 percent attacks could be attempted by leveraging a combination of high hash rate and high network connectivity (thus requiring less than 50 percent of the hash rate).

Secondly, when the block generation rate is raised too high, the selfish mining strategy highlighted by Eyal and Sirer (as discussed in a previous QA), can be employed by weaker miners. This is also a serious security vulnerability.

Both of these issues are not addressed by the GHOST protocol in the original whitepaper. In fact, the original whitepaper does however say that these issues will be addressed in a companion paper that was published in 2015. However, when this companion paper is examined they completely abandon the idea of blockchains opting instead for a DAG-like protocol known as "Inclusive" (a derivative of which was subsequently used for Ethereum). As such, GHOST alone definitely does not really address these issues other than to propose a completely different structure in a separate paper.

Therefore, we can confidently state than GHOST alone does not simply allow the difficulty to speed up block generation for arbitrarily high transaction throughputs.

It is nonetheless interesting to note that Vitalik Buterin, the main Ethereum creator, has hinted at a possible "million transaction per second" scaling involving the use of a two-layer scaling solution combining both Sharding and Plasma in synergy.

Problem 4

In Conflux, is it possible for an attacker to create a malicious block choosing a very early block as its parent block to disrupt the transaction total order? Why?

Solution

No. Indeed, as detailed in the official whitepaper by Prof. Long and others, Conflux was designed to deterministically drive a transaction total order in the blockchain ledger. Simply put, Conflux delays the transaction total ordering to optimally process concurrent transactions and blocks. This is justified by the fact that transactions are rarely in conflict in blockchains. However, we must examine the protocol in more depth, under its existing assumptions, to be able to determine if it is possible for an attacker to create a malicious block from a very early block to disrupt transaction total order.

More specifically, each node that wants to issue a transaction broadcasts it on the gossip network. Each other node in turn adds that transaction to their transaction pool and is in charge of totally ordering a group of multiple transactions which eventually get packed into a block (using PoW) and broadcast via the gossip network. Each node in turn receiving a block will remove those transactions from their transaction pool. Since transactions may arrive at different times, the nodes in the network are likely to have slightly different DAGs so concurrent blocks might pack duplicate or conflicting transactions. This is resolved by the consensus generator.

In Conflux, new blocks identify their parent block via a single parent edge (a bit like in blockchain, this represents a vote for that block from the child). In addition, reference edges are used to show that a particular block is generated before another (the edge is directed from the newer block to the older block). This allows the DAG to be divided into epochs where the epoch is named according to the most recent block on the pivot chain in that epoch. The pivot chain is then selected using the GHOST rule.

From both this pivot chain structure and the epoch method, the Conflux algorithm first sorts both the block order which in turns allows it to sort the transaction total order. Conflux sorts the blocks in topological order within their corresponding epochs. If two blocks in an epoch have no partial order relationship, Conflux breaks the ties deterministically using the unique IDs of those blocks. Then the transactions are ordered based on the total orders of enclosing blocks. If two transactions are in the same block, Conflux sorts them based on the appearance order in the block. These transactions are then checked with the rest of the blocks. If transactions appear in multiple blocks, Conflux only keeps the first appearance and discards the redundant ones. Similarly, if two transactions are conflicting with each other, Conflux simply discards the second one, avoiding overspending or double spending.

Therefore, we can conclusively state that no, choosing a very early block as parent block is not able to disrupt the transaction total order as it would be locked into an epoch by the Conflux protocol. Indeed, as detailed in the Conflux whitepaper, a block attempting to do this will have no children so will not become the pivot chain block. Thus, the attacking block will have to wait for references from future pivot chain blocks so will always belong to a future epoch even if choosing an earlier block as parent (since the pivot chain is protected by Nakamoto consensus via its weight).

Problem 5

How much hash power an attacker must control for him/her to deterministically launch double spending attacks in Conflux? Why?

Solution

The hash power needed for an attacker to deterministically launch double spending attacks in conflux is 51%. This is because, as discussed previously, attackers cannot bypass the epoch-based DAG to disrupt order. Hence, the attackers only option to launch a double spending is to try to overpower the pivot chain. Since this is protected by Nakamoto consensus (see question 4), 51% hashing power will be required to deterministically pivot onto a block that the attacker desires in order to launch double spending (i.e. join the attackers block to the pivot chain).

Indeed, this is confirmed by figure 9 in the Conflux whitepaper which shows that, at an attacking hash power of 40%, a user could still wait 2500s to get a confirmation with 0.01% risk tolerance.

Problem 6

Is it possible that the Algorand sortition algorithm selects a committee that contains a majority of malicious nodes? If so, is this going to be a problem?

Solution

It is possible for Algorands sortition algorithm to select a numerical majority of malicious nodes (i.e. a larger number of malicious nodes than of honest nodes), however this is not a long-term problem.

This is because Algorand is not a Proof-of-Work (PoW) consensus protocol. Instead it relies on Byzantine Fault Tolerance (BFT) and a form of pure Proof-of-Stake (PoS) to achieve node agreement and consistency on the next block. Indeed, this allows the blockchain to be linear (i.e. have very little probability of forking 10^{-18}). One of the ways that this is implemented is through the committee selection. In short, a committee is chosen amongst all the users from which leaders are selected. These leaders propose the next block and the rest of the committee achieves Byzantine agreement on the next block.

Indeed, in Algorand a committee is chosen randomly among all users (or nodes) based on each users weight. The user weight is assigned based on how much stake they have of the Algorand cryptocurrency in their account (inspired from the PoS consensus algorithm). The assumption being that if you have a large stake in the cryptocurrency, you would not want to subvert the protocol and destroy its future. Thus, by choosing weighted users according to their stake, Algorand ensures a sufficient weighted fraction of the committee members are honest enough (or dishonest but unwilling to act). A leader is chosen from this committee which could be malicious as some malicious users could be in the committee as previously explained but a large weight due to large investment would be the variable biasing the probability of selection.

Therefore, this is not a problem in Algorand. This is because even if a majority of malicious nodes get into the committee they will only have detrimental decision-making power if they have a large amount of stake in Algorand. Indeed, the whitepaper states that as long as the attacker controls less than one third of the monetary value (main assumption behind most of the mathematical proofs), Algorand can guarantee that the possibility of double spending or forks is negligible (around 1 chance in 10^{18} meaning it would take about as long as the age of the universe). Therefore, this is not going to be a problem.

Problem 7

The random seed of the sortition algorithm is critical for the security of the Algorand. Suppose an attacker now can control the starting seed of the Algorand sortition algorithm. Explain an attack strategy for the attacker to subvert the Algorand blockchain.

Solution

If the attacker can control the starting seed of the Algorand sortition algorithm, he may be able to choose a seed that favours the selection of corrupted users within the committee.

This in turn would mean that you could frequently end up with a committee of many corrupted users with disproportionate cumulative priority which would make Algorands assumption of only one third of malicious nodes (by stake) slightly riskier. Indeed, if this critical threshold was passed, the attacker would be able to fully break the Algorand protocol. This is because the next seed is calculated by input from the committee so the malicious committee would be able to repeatedly dominate and destroy the Algorand protocol.

However, if the threshold is not passed but we get consistently more prioritised corrupted users in the committee, the attacker could cause adverse effects which would damage the Algorand protocol. Indeed, more prioritised corrupted users in the committee means a higher likelihood that corrupted users end up with disproportionately high priority. If this is the case, and if a large number of malicious users submit malicious blocks, this would lead to an increase in the number of empty blocks generated to counter this. This is because, if the nodes cannot agree on a proposed block, they will agree on an empty block. This increasingly large number of empty blocks would prevent any real transactions from being confirmed for an increasingly large number of rounds.

As we can see, both total protocol destruction and this pollution attack would be some of the multiple methods for the attacker to subvert the Algorand blockchain if the random seed rule was not enforced properly.

Problem 8

In Algorithm 8, why the algorithm only considers the consensus "final" when it reaches the consensus at the very first step ($step == 1$)?

Solution

Byzantine Agreement (BA*) executes in steps where the gossip network agrees in a Byzantine-safe method on a new block. To do this, two types of consensus are used: final consensus and tentative consensus.

If a user reaches final consensus, it means any other user that reaches final or tentative consensus in that round must agree on that same block value as this user who has reached final consensus. Therefore, Algorand only confirms transactions once the block reaches final consensus.

If a user reaches tentative consensus, it means that other users may have reached tentative consensus on a different block and that no user has reached final consensus yet. Users confirm transactions from tentative blocks only if and when a subsequent block reaches final consensus. BA* produces tentative consensus if an attacking network proposes that a malicious block is the next block but this is quickly reversed once eventually an honest block is agreed upon in subsequent blocks. This is because there is an upper bound on how long malicious nodes can keep their malicious fork. When there is temporarily no consensus an empty block is the resultant output. Since BA* is periodically invoked in Algorand, consensus is eventually achieved on which fork should be used going forward, hence the need for tentative consensus.

Now, we can see in algorithm 8, the combination of all these elements of the Algorand protocol. However, algorithm 8 only works in the case of a network with weak synchrony if we consider the consensus final only when it reaches the very first step.

This is because if our network has weak synchrony (i.e. there is a partition), BinaryBA* could return consensus on different blocks. For example, suppose in a first stage all users could vote for a particular block but only one user receives those votes. And say the users vote again because of a timeout but those votes get lost by a network drop. This situation could only be recovered from by introducing the aforementioned concepts of final and tentative consensus. Indeed, the tentative consensus here is used to express that BA* is unable to guarantee safety either because of a network asynchrony or because of a malicious block. Therefore, the only way we can confidently have consensus and guarantee safety (by guaranteeing an agreement on the block) is by only considering the consensus as final in the first step.

Therefore, the algorithm only considers consensus final when it reaches the consensus at the very first step in order to guarantee safety under all conditions, even in the case of a network dropping out while being weakly synchronous.

Problem 9

In Algorithm 8, why the algorithm needs a CommonCoin mechanism?

Solution

The CommonCoin mechanism is used to generate an unpredictable Boolean value that most nodes will agree on. Indeed, this CommonCoin mechanism is used to prevent a particular type of attack that would lead to a malicious user causing the system to go into an undesirable state.

More specifically, consensus in algorithm 8 could theoretically get stuck if honest users are split into two groups. If both groups vote for different values, neither group is large enough to gather enough votes alone and so need to rely on the attackers votes. In this case, the attacker can deterministically predict what vote each user will vote for in the next step. Therefore, since the attacker knows this he can either send his vote or not just before the timeout to indefinitely split the groups. This could violate the liveness properties of Algorand by causing perpetually fluctuating votes without every reaching consensus.

To restore liveness, even in case of this attack the clever trick of the CommonCoin is implemented. This CommonCoin tells the users to vote one way or another in a random non-deterministic way to remove the stranglehold that the attacker has. On each iteration the attacker will only have a 50/50 chance of correctly predicting the future so the probability of consensus will rapidly converge to 1 restoring liveness.

Indeed, it is critical in most concurrent systems to preserve the property of liveness. Liveness is a necessary condition for a concurrent system to continue to make progress, thus preventing deadlock issues from arising. Indeed, if the CommonCoin method was not implemented, this could result in an infinite while loop constantly running without making any progress leading to livelock.

In cryptocurrencies, such states of deadlock could mean that all tokens in the blockchain become immovable and frozen. This would instantaneously render the token worthless and would result in the disastrous loss of assets of every single user without possible salvation. The only option would be to rebuild a new mirror blockchain with fixed code and with a patch where the coin is pre-mined such that all previous users have the same assets as they did in the deadlocked currency. The transfer of all these private keys and assets from one chain to another would most likely be a logistical and untrustworthy mess, therefore seeming unfeasible in practice. It is for this reason that deadlocks must be avoided at all costs (and not just in cryptocurrencies).

Problem 10

In Algorithm 8, why the algorithm terminates after MAXSTEPS? What would happen if the algorithm keeps trying without a MAXSTEP limit? Whats the security consequence here?

Solution

In Algorithm 8, the while loop has a terminating limit set by MAXSTEPS. Following this terminating MAXSTEPS limit, Algorand proceeds onto a recovery routine.

The reason this was implemented is because of a particular situation that arises since BA* relies on a committee to declare final consensus rather than relying on all participants. The situation is that even if a user observes final consensus, a malicious node that controls the network could be able to repeatedly prevent other users from reaching any kind of consensus. Indeed, if this is allowed to happen for an arbitrary number of steps, each additional step gives the adversary a small probability of reaching consensus on a different value.

Therefore, to avoid this unwanted behaviour and to bound the total probability of an adversary reaching an unwanted value, BA* puts a limit on the total number of allowed steps (as shown in algorithm 8).

Indeed, if the algorithm runs over these number of steps it will default back to a safe recovery routine rather than giving further chances to the malicious node.

Problem 11

Why Algorand uses a more complicated BFT algorithm that may not always reach final consensus? Why not Algorand runs the standard PBFT algorithm among selected committee members? (Hint: Read the last part of the introduction of the paper)

Solution

Algorand indeed uses a more complicated BFT algorithm instead of the standard PBFT between its elected committee members. This is done for two main reasons. The first reason is to improve the performance of PBFT and make it more suitable for cryptocurrencies via 2 main changes. The second reason is to improve security and again 2 main changes are made to achieve this (for a total of 4 major changes to PBFT).

The first change made to improve performance is done by reducing the overheads present in PBFT. Prof. Silvio Micali (MIT) who was behind much of the work in developing Algorand, has heavily criticised the original PBFT algorithm in some of his live talks. He argues it is incredibly inefficient with many stages and many message overheads, as we have shown in previous homeworks. Indeed, this leads to PBFT being unusably slow at scale and impractical for direct use in cryptocurrency implementations. He improves on this by decreasing the size of the messages and reducing the complex number of layers to a simpler structure.

The second change made to improve performance has to do with PBFTs assumption of a fixed number of servers. Since this is not applicable for cryptocurrencies, Prof. Micali suggested changes into a more flexible and more suitable BFT algorithm.

Now for the next two major BFT complications, these address security improvements. These changes were made to avoid byzantine nodes from attacking the leading node using targeted DoS attacks, targeted compromises or similar strategies to attempt to compromise the network.

In the first security-related BFT complication, the new leader is elected in a secret way using a verifiable random function (VRF). This verifiable random function is used by each node along with their private key to find out if they are the new leader. Thus, only the selected node will know they have been selected. This means no malicious user would be able to target the leader before he is announced.

In the second security-related BFT complication, the leader is changed after each new announcement. So once the new leader secretly finds out his role, he can send out his proof of leadership as well as his block suggestion before a new leader is chosen. This means no malicious user would be able to target the leader after he is announced either.

Therefore, we have shown why Algorand adds a total of 4 main complications to the PBFT method in order to achieve better performance and a higher level of security for protection against targeted compromises or targeted DoS, even if this may not always reach final consensus. Indeed, BA* executes in repeated steps and it is possible for only tentative consensus to be met at a point in time. However, given the assumptions made in the paper, eventually final consensus will be reached for the next block which is all that matters for the ultimate operation of Algorand.