Lecture 14: Blockchain protocols with finality: Streamlet and HotStuff

Principles of Blockchains, University of Illinois,
Professor: Pramod Viswanath
Scribe: Xuechao Wang

March 16, 2021

Abstract

So far, we have seen the longest chain protocol (module 1) and its variants (module 2). These protocols all favor liveness, but are not safe under network partition. In this lecture (as a beginning of module 3), we study a different family of blockchain protocol called Byzantine fault tolerant (BFT) protocols, which guarantee deterministic safety (aka. finality). We study two BFT protocols, Streamlet and HotStuff, the former is simple, and the latter has state-of-art performance.

Introduction

So far we have focused on the longest chain consensus protocol (module 1) and modifications to scale performance (module 2). The major advantage of the protocol is its very strong liveness property: the protocol is live even with minuscule honest hash power: a single honest miner has a chance to succeed in the PoW lottery and get to propose a block, thus extending the longest chain regardless of the network synchrony. The protocol guarantees security for a majority honest hash power, but with two caveats:

- *Probabilistic Safety Guarantee*: the safety of the confirmed blocks is provided in terms of the *probability* of deconfirmation ("error").
- *Synchronous Network Operation*: honest nodes crucially need to synchronize at periodic intervals on the state of the longest chain, so that they can work on extending the common longest chain, avoiding forking.

In summary:

> The longest chain protocol prioritizes liveness over safety.

In some applications, safety is very important, especially deterministic guarantees (known as "finality") for a confirmed ledger entry, e.g., financial applications. In such a case, there is not much respite the longest chain protocol (and any of the modifications we saw in the past module) will offer. In this lecture, we will study blockchain protocols that offer finality, and even if the network is not synchronous. This is the goal of this lecture. The protocols we see are the culmination of a sequence of works in a family of BFT (Byzantine fault tolerant) protocols, studied in computer science since the 1980s.

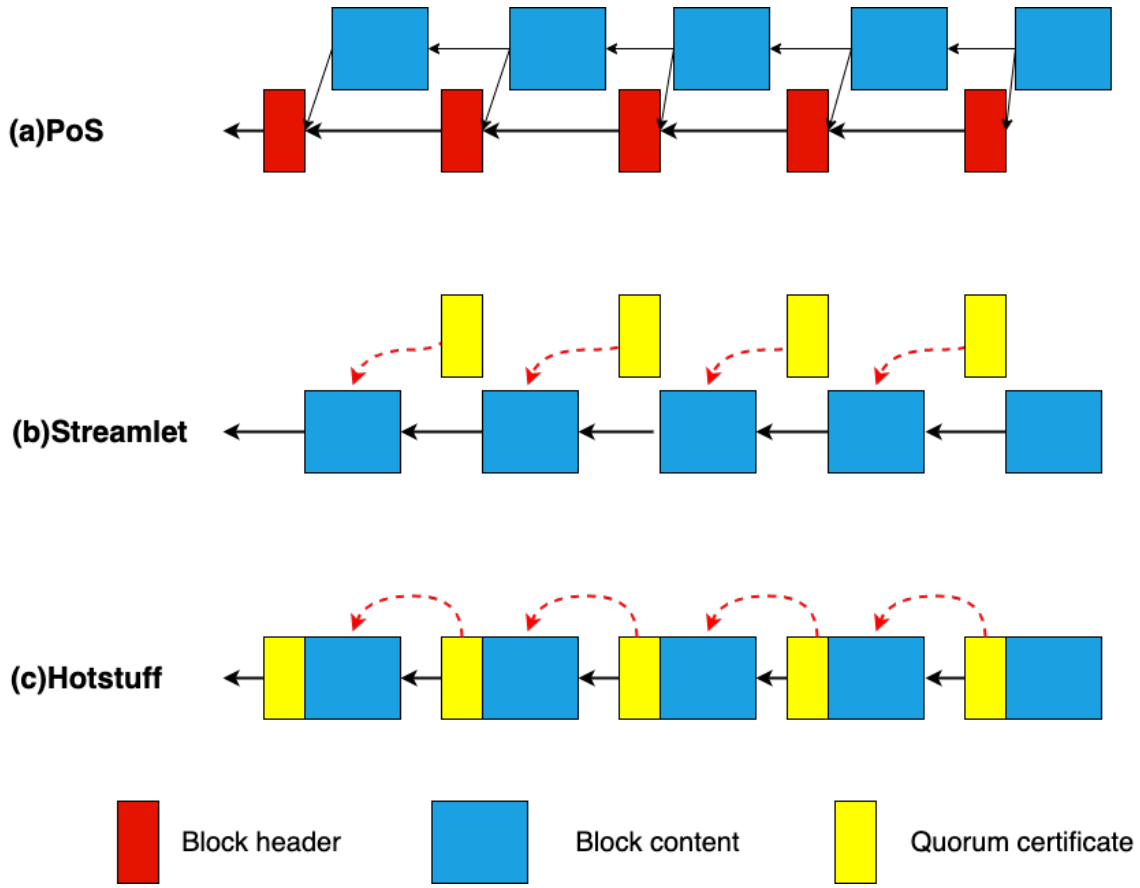

Figure 1: (a)Proof-of-stake longest chain protocol from Lecture 12; (b)Streamlet;(c)HotStuff.

BFT Protocol Setup

We instantiate the BFT protocol with a *fixed* set of participating nodes (referred by N). The participating nodes have a fixed identity (public key), known to all participants. This is a departure from the permissionless blockchains supported by the longest chain protocol; we will discuss how to adapt the BFT protocols to the permissionless setting next lecture.

The protocol proceeds in *rounds*, denoted by integers, much like the PoS longest chain protocols we studied in Lecture 12. Unlike the PoS protocols, here a *single* proposer is elected in each round. The proposer should be chosen and verified in a distributed manner. In a permissioned system, we can do the proposer election in two simple ways: (a) A round robin fashion, e.g., round i's proposer is the node (i mod N); (b) Using a distributed pseudo-random function or a hash function $H : \{0,1\} \to [N]$, round r's proposer is computed as $H(r)$.

Streamlet

Streamlet protocol

Chain rule. Streamlet protocol works as follows. In every round:

- The round's designated proposer proposes a new block extending from the longest notarized

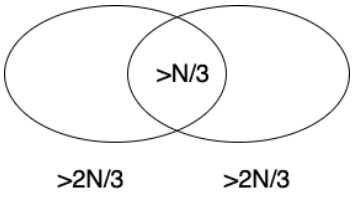

Figure 2: The intersection of two quorums must be greater than $N/3$.

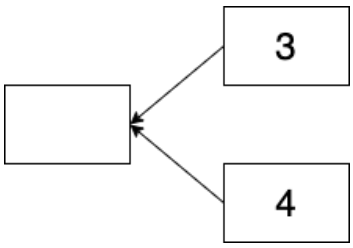

Figure 3: Two blocks at the same height can be both notarized.

chain it has seen (if there are multiple, break ties arbitrarily). The notion of "notarization" is defined below.

- Every node votes for the first proposal they see from the round's proposer, as long as the proposed block extends from (one of) the longest notarized chain(s) that the voter has seen. A vote is a signature on the proposed block.
- When a block gains votes from more than $2N/3$ distinct nodes, it becomes notarized. A chain is notarized if its constituent blocks are all notarized.

We call the set of distinct votes on a notarized block a *quorum certificate* (QC). This choice of quorum size to be more than $\frac{2N}{3}$ makes sure that at most one block per round can be notarized as long as the number of malicious nodes is less than $N/3$. This is because the intersection of two quorums has size at least $N/3$ (see Figure 2) and only malicious nodes will potentially equivocate.

Confirmation rule. Since at most one block per round can be notarized, can we simply confirm notarized blocks? Unfortunately this is insecure, because two blocks in different rounds can be both notarized at the same height of the blockchain as seen by the following example.

Confirming a 1-deep notarized block is insecure. Let $f < N/3$ be the number of malicious nodes. In Figure 3, a malicious proposer from round 3 proposes block B_3, but it only sends B_3 to x honest nodes, where $2N/3 - f < x < 2N/3$. If all f malicious nodes keep their votes on B_3 private, then B_3 will not be notarized in the view of any honest node. Suppose the proposer of round 4 is honest, then it will propose a block B_4 at the same height as B_3. Every honest node votes for B_4 , and it gets notarized. After that, all f malicious nodes release their votes on B_3 and make B_3 notarized. Since B_3 and B_4 are conflicting (not on a chain), confirming either one of them will lead to a safety violation. So notarization does not mean confirmation.

Confirming a k-deep notarized block is also insecure. In the longest chain protocol, we have also seen that confirming the tip of the chain is not secure. The solution in the longest chain protocol is to confirm according to the k-deep rule, with sufficiently large k. The natural question is whether the k-deep confirmation rule (confirming the k-deep block in the longest notarized chain) is secure in Streamlet? Unfortunately, this is not the case: during asynchrony, the adversary can play the trick in the previous attack (Figure 3) at every height to create two equally long chains of arbitrary length. Recall that we assume the adversary can delay each message arbitrarily when the network

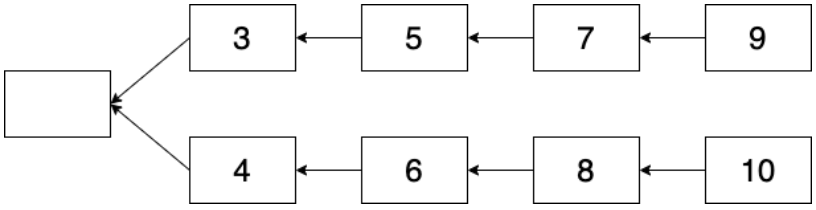

Figure 4: A balance attack.

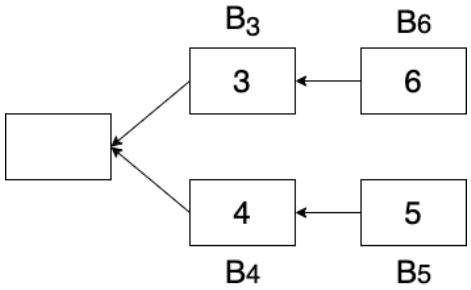

Figure 5: Two blocks with consecutive round numbers are not enough for secure confirmation.

is asynchronous. In Figure 4, at each height, the adversary makes sure that only x honest nodes vote for the block (with an odd round number) in the chain above, where $2N/3 - f < x < 2N/3$. And after the conflicting block (with an even round number) in the chain below gets notarized, the adversary releases f private votes to make the block above notarized. Because of this *balance attack*, the k-deep rule is still not safe in Streamlet.

Confirming a 2-deep notarized block with consecutive round numbers is insecure. Note that in this balance attack, there are no blocks with consecutive round numbers in a chain. Then how about confirming the notarized chain up to the second last block when there are two adjacent blocks with consecutive round numbers at the tip? Note that we cannot confirm the last block in the chain because of the attack in Figure 3. With this confirmation rule, no block is confirmed in Figure 4, so there is no safety violation for this attack. However, a different attack is fatal; see Figure 5. Here the adversary conducts an analogous attack on blocks B_3 and B_4 to make them both notarized. A malicious proposer from round 5 places the block B_5 as a child of B_4, and makes sure that only x honest nodes vote for B_5, where $2N/3 - f < x < 2N/3$. B_6 is proposed as a child of B_3 and every honest node will vote for it. After B_6 is notarized, all f malicious nodes release their votes on B_5 and make B_5 notarized. In this case, we see that B_4 and B_5 are notarized as a chain; so B_4 is in a 2-deep notarized chain with consecutive round numbers and will be confirmed. However confirming B_4 will lead to a safety violation, because some honest nodes may adopt B_3 and B_6 as the longest notarized chain.

Confirming a 3-deep notarized block with consecutive round numbers is secure. Since the rule with two blocks in a row does not work, let us try one more time: *three blocks in a row*. If in any notarized chain, there are three adjacent blocks with consecutive round numbers, we confirm the prefix of the chain up to the second of the three blocks. It turns out this one is safe! Suppose one honest node sees three notarized blocks B_5, B_6, B_7 from rounds $5, 6, 7$ in a chain (see Figure 6), to prove safety we argue that no block can be notarized at the same height as B_6. For the contradiction, suppose there is a notarized block B from round X conflicting B_6. Due to the fact that at most one block can be notarized per round, we know $X > 7$ or $X < 5$.

- **Case 1:** $X < 5$. Since block B is notarized, it means that more than $N/3$ honest nodes,

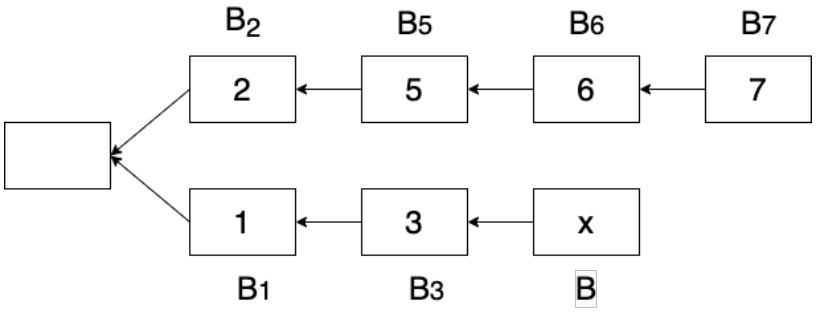

Figure 6: Streamlet confirmation example. In this example, the prefix of the top chain up to the round-6 block is considered confirmed.

denoted by the set S, voted for block B and not only so, at the time of the voting (that is, during round $X < 5$), they must have observed block B_3 notarized. Now the honest nodes in S will not vote for block B_5 during round 5, since it fails to extend a longest notarized chain seen, which is block B_3 or longer. Since $f < N/3$, this means that block B_5 can never get notarized in an honest node's view, which leads to a contradiction.

- **Case 2:** $X > 7$. Since block B_7 is notarized, more than $N/3$ honest nodes (denoted the set S) must have seen a notarized block B_6 by the time they vote for block B_7 (i.e., by the end of round 7). As a result, in round $X > 7$, the set S of nodes must have seen block B_6 notarized and will not vote for block B, since block B now fails to extend the longest notarized chain seen (which is block B_6 or longer). Since $f < N/3$, this means that block B can never get notarized in an honest node's view, which leads to a contradiction.

Therefore, we can conclude that this three-block confirmation rule is safe. Note that the above proof holds, no matter what the actual network delays are, and even if honest nodes' rounds are not synchronized (as long as the local round numbers are monotonically increasing). Of course, if the network is partitioned and honest nodes' messages are being held up, then we cannot guarantee progress. As explained next, liveness ensues in Streamlet during "periods of synchrony", i.e., during a period of time in which messages between honest nodes are delivered within a known bound Δ.

If we set the round duration to be the maximum round-trip delay (2Δ), then when an honest proposer proposes its block at the beginning of a round. This should allow sufficient time for all the honest nodes to get their votes in and notarize the block. Therefore, if the adversary has less than 1/3 of all nodes, it does not have the power to stop chain growth. Formally it can be shown that once there are 5 consecutive rounds of honest proposers, a new honest block will be confirmed.

Performance of Streamlet

Communication complexity of Streamlet. Streamlet requires nodes to *echo* received messages (blocks or transactions) to everyone else. Let the block size be B bits and vote size by V bits. In each round, there are N senders, N receivers, and on each link one block and N votes (due to echoing) are sent. The total communication adds up to $N^2(B + NV) = N^2B + N^3V$ bits per block, even in the case of an honest proposer. If N is large (i.e., we want the protocol to be scalable), N^3V is the dominating term in the communication complexity. Furthermore, Streamlet requires echoing for its security guarantees (thus necessarily incurring N^3 communication complexity); this is best seen via the example below (see Figure 7).

Consider a blockchain system with $N = 7$ nodes $\{a, b, c, d, e, x, y\}$; suppose nodes x and y are Byzantine. By round r, all honest nodes share the same view of the world. Then, x is chosen to be the proposer of round r. During the first half Δ of round r it sends its block proposal B only to

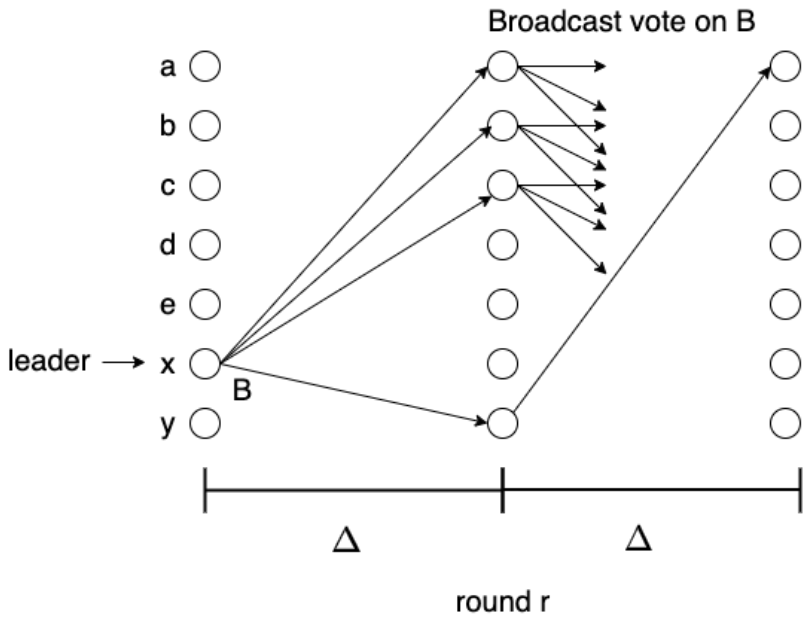

Figure 7: Liveness is compromised if there is no message echoing in Streamlet.

nodes a, b, c and y. In the second half of the round, a, b and c, follow the protocol and vote for B, sending their signature to all other nodes. The other Byzantine node, y, on the other hand, sends its vote only to a. At this point the round ends and no further votes on B will be cast. It is clear that at this point a has collected 5 ($> 2N/3$) votes for B (from a, b, c, x, y) while any other honest node collected only 4 of them. This leads to B being notarized only by a. If there are no vote echoes, in the following rounds a does not vote for any proposal that does not extend B. If a and y cease to participate, liveness is compromised. When a becomes the leader, it proposes to extend B but no one else votes for it, because they don't know it is notarized. Therefore, the chain stops growing.

Since we have to use at least $N - 1$ messages to spread the block among all nodes, the best communication cost we can expect would be $O(N)$ per block. We say a blockchain protocol achieves *linearity* if the communication complexity is linear in the number of nodes. Obviously, Streamlet is far from achieving linearity due to broadcasting and echoing.

Latency of Streamlet. To guarantee liveness, Streamlet makes an assumption that rounds operate in lock-step with 2Δ duration. Under the "normal path" (i.e., no adversary and no forking), the confirmation latency of Streamlet is two rounds (4Δ in time duration). If there are $f < N/3$ malicious nodes, Streamlet still guarantees liveness whenever there are 5 consecutive honest proposers. This happens on an average once every $1/(2/3)^5 \approx 7.6$ rounds, about 15Δ. Recall that Bitcoin latency is $O(\log_e(\frac{1}{\epsilon})\frac{1}{\lambda\Delta})\Delta$, where ϵ is the confirmation error probability and $\lambda\Delta \ll 1$ for security. So compared to Bitcoin, Streamlet achieves much better latency: small constant and finality (i.e., deterministic confirmation).

However, BFT protocols can actually achieve even better latency by having non lock-step rounds. Streamlet foregoes an important property of asynchronous consensus protocols, called *responsivity*: the ability to advance at the speed of the actual network delays without waiting for maximal network delays.

External client support. In Streamlet, the votes are not recorded on the blockchain itself. Therefore, while Streamlet creates a total-order on transactions, the question of how an external client (who is not in the permissioned committee) can verify the correctness of the ledger and confirm a transaction is not addressed.

HotStuff

HotStuff is the consensus protocol behind Facebook's Diem project. HotStuff is very similar to Streamlet, but it achieves both linearity and responsiveness. Now that we have seen Streamlet in detail, it is straightforward to extend the protocol into HotStuff.

HotStuff highest QC. In HotStuff, a block is linked to its parent using the QC itself (on-chain), so everyone can verify it has been notarized (see Figure 1(c)). Every node keeps the QC with the highest round number (HighQC) it knows of. As in Streamlet, a proposer proposes a block extending highQC, while a node votes for a proposal if it extends the branch of its highQC. But an important difference is that a node only sends the vote to the proposer of the next round. Therefore, in HotStuff, the communication cost per block is $NB + NV$; thus HotStuff has linear complexity.

HotStuff finalization rule is the same as in Streamlet: whenever three alternating blocks whose rounds are consecutive are formed, the middle block of the three blocks becomes finalized (along with all blocks in the chain to the genesis).

HotStuff round synchronization. HotStuff does not require round synchronization. A proposer can propose a new block immediately after it receives a new QC, i.e., the actions in HotStuff are event driven rather than time driven as in Streamlet. The protocol can make progress at the speed of the actual network delays, so HotStuff is responsive. By doubling the time each node spends in each round until a decision is made, liveness is guaranteed.

Protocol Forensics

In the Streamlet and HotStuff protocols we have seen that there is a fixed number of participants and their identities (through their public keys) are fixed and known as well. Can these identities be used to provide some incentives for honest behavior? In particular, if the nodes that collude to launch a safety attack can be identified with cryptographic integrity, such *forensics* would provide a strong disincentive to malicious behavior. Indeed, a single honest node can detect at least $\frac{N}{3} + 1$ malicious actors (without a need to collaborate with any other node), for both the Streamlet and HotStuff blockchain protocols.

Suppose an honest node sees a safety breach. Then there must be more than $\frac{N}{3}$ nodes, each of which must have either voted on two blocks with the same round number or voted on two blocks with depth and round number ordering reversed; see Figure 8. Both of these conditions are observable by the honest node; further an intersection of the QC of the conflicting blocks yields cryptographic evidence of the malicious actors.

Conclusion

In this lecture, we have upgraded probabilistic safety in the longest chain protocol to deterministic safety (known as *finality*); but this improvement has come at two costs:

- there is a fixed number of participants and their identities (through their public keys) are fixed and known as well, which does not suit the permissionless nature of blockchains we have discussed so far. In the next lecture (Lecture 15) we study a blockchain protocol, also based on the BFT family, that suits a permissionless architecture and is robust to adaptive adversaries just like the PoW longest chain protocol.
- liveness is weakened in the pursuit of strengthening security guarantee to a finality: substantial honest participation is needed now, unlike the longest chain protocol that allowed dynamic participation and even a single honest miner ensured liveness. How to have both finality (deterministic safety) and dynamic availability (liveness even under varying participation levels of honest nodes) is the topic of the lecture after next (Lecture 16).

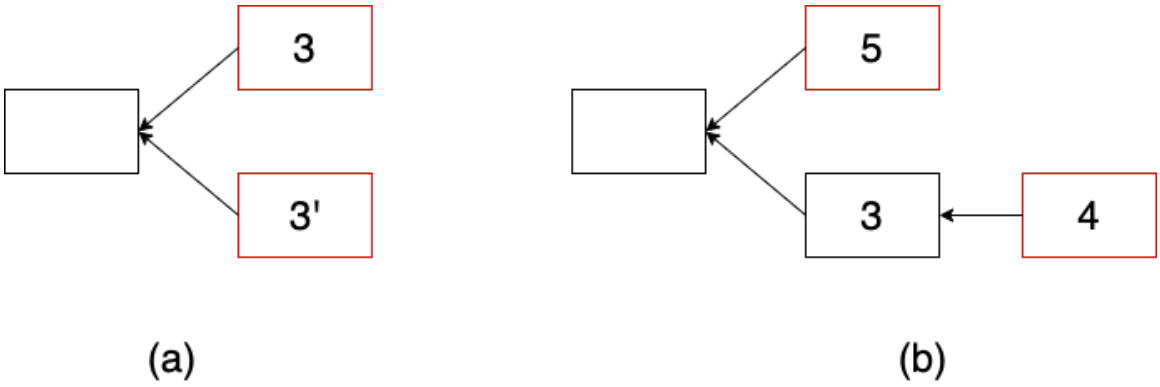

Figure 8: Safety breach in Streamlet/HotStuff. When an honest node observes these two conflicting notarized blocks (in red), at least $N/3+1$ malicious nodes can be detected by taking the intersection of the corresponding two QCs.

Reference

We have presented HotStuff as an optimized version of Streamlet. We present them in this order because Streamlet is simpler, although HotStuff predates Streamlet by a few years. Our presentation is inspired by a blog post, co-authored by Dr. Dahlia Malkhi (one of the inventors of HotStuff); the blog post compares Streamlet and HotStuff sharply and presents a clean way to morph Streamlet back into HotStuff.