Lecture 9: Scaling Latency

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

February 23, 2021

Abstract

In this lecture, we see that there is a fundamental tradeoff between latency and security in Bitcoin. We study the Prism protocol, a simple variant of which was introduced in the previous lecture, which further separates voter blocks from proposer blocks. By having multiple parallel chains for voting, Prism decouples latency from security and achieves fast confirmation.

Bitcoin Latency

In Bitcoin, when a transaction has been included in a block, it will be confirmed only after the block is buried k-deep in the longest chain. Furthermore, to ensure confirmation with high probability, k must be large. The average latency of confirmation is then the product of k and average inter-block arrival time; again just like the throughput, the latency in Bitcoin is also limited by security. How bad is this tradeoff? We study this next.

As in the analysis of throughput, our analysis starts with the case when the network delay is zero, i.e., $\Delta = 0$. In this case, the largest probability of deconfirmation ("error probability") ε is achieved by the private attack (see Lecture 6). Further, we have seen that the error probability decays exponentially in the depth k of the confirmation rule:

$$\varepsilon = e^{-ck}, \qquad c = -\log_e(4\beta(1-\beta));$$

here β is the fraction of the hash power of the adversary. Suppose that the target difficulty of the mining puzzle is set such that the average time to succeed in mining a block is $\frac{1}{\lambda}$. Then we know that themaximum average inter-block time is $1/(1-\beta)\lambda$, which is met when the adversary does not participate in mining (see Lecture 8). Therefore, the latency of Bitcoin is given by

$$\tau_{LC} = \frac{k}{(1-\beta)\lambda} = \frac{1}{c(1-\beta)\lambda}\log_e(\frac{1}{\varepsilon}) = O(\frac{1}{\lambda}\log_e(\frac{1}{\varepsilon})),$$

where the constant hidden in the $O(\cdot)$ only depends on β. We can see that the latency depends on the error probability ε, i.e., security. For example, we can compute that $\tau_{LC} \approx 66/\lambda$ when $\beta = 0.3$ even for a modest setting of $\varepsilon = 10^{-3}$; Bitcoin has $\frac{1}{\lambda}$ of 10 minutes, so the latency works out to 11 hours! We plot the tradeoff between latency and security of Bitcoin against the private attack (which is the worst-case attack for $\Delta = 0$) in Figure 1; indeed the latency is terrible even for modest requirements on the security parameter ε.

For the general case $\Delta > 0$, we have seen in Lecture 6 that the private attack is no longer the worst case attack in terms of success probability. Nevertheless, the private attack still provides a *lower bound* on the error probability and a corresponding lower bound on the latency for a given error probability. Both these quantities only get worse as Δ increases, due to the resulting forking. From

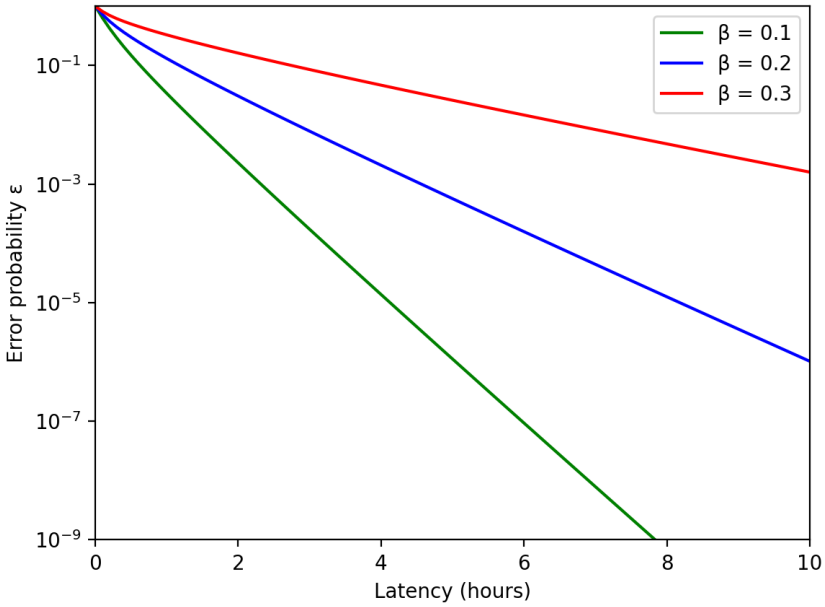

Figure 1: Bitcoin's latency–security tradeoff for different β assuming $\Delta = 0$. We set λ to be 6 blocks per hour, as in Bitcoin.

a practical point of view, *upper bounds* on the error probability (and latency) are most interesting; an upper bound on latency provides practical guidance for a user of Bitcoin when to consider a transaction confirmed with a desired level of confidence. A recent work reports such upper bounds which are fairly close to the lower bounds implied by the private attack. For example, when the adversary controls 10% of the total mining power and the block propagation delays are within $\Delta = 10$ seconds, a Bitcoin block can be confirmed with less than 10^{-3} error probability after 5 hours 20 minutes, or with less than 10^{-10} error probability after 12 hours 15 minutes. Figure 2 illustrates the security-latency tradeoff as implied by the lower and upper bounds.

Prism: Fast Confirmation

Lecture 8 introduced Prism 1.0 with two types of blocks: *transaction* blocks and *proposer* blocks. These two blocks decouple the security and payload (data) aspects of the longest chain protocol. Within the security aspect (embodied by the proposer block in Prism 1.0) there is a further subdivision of two properties. First, the block *proposes*, i.e., acts as a leader for the "round". Second, the block adds confidence to the ancestor blocks along the chain to the genesis (as embodied by the k-deep confirmation rule), i.e., acts as a "voter" to the ancestor blocks. Each block votes for all ancestor blocks through the parent link relationships. Continuing the deconstruction principle in Prism 1.0, we can further decouple the role of voting from the proposer blocks by having separate voter blocks. But how to organize the voter blocks? The full Prism protocol answers this question.

Just as in Prism 1.0, the *proposer* blocktree in Prism anchors the blockchain. Each proposer block contains a list of reference links to *transaction* blocks that contain transactions, as well as a single reference to a parent proposer block. Honest nodes mine proposer blocks following the longest chain

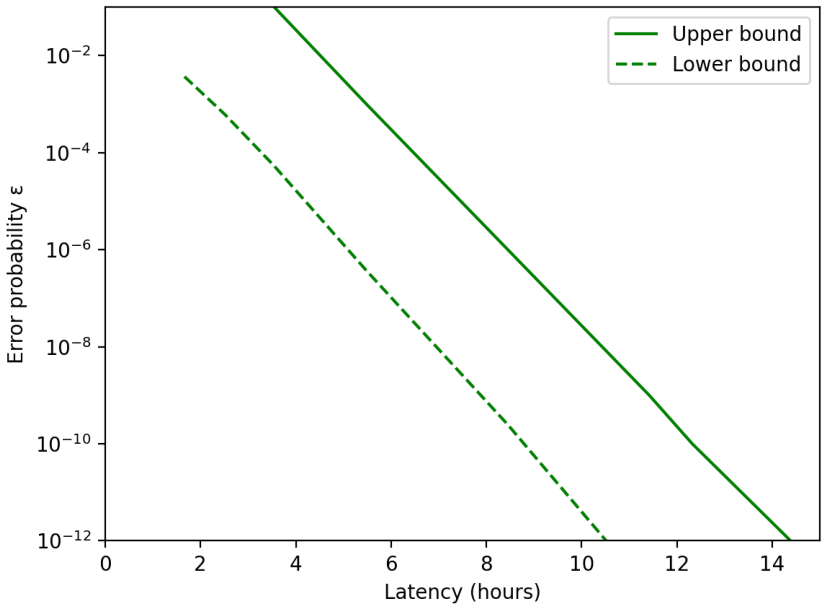

Figure 2: Bitcoin's latency–security tradeoff for $\beta = 0.1$ when $\Delta > 0$. We set λ to be 6 blocks per hour as in Bitcoin and $\Delta = 10$ seconds. The lower bound on latency is derived from the private attack, while the upper bound is borrowed from this work.

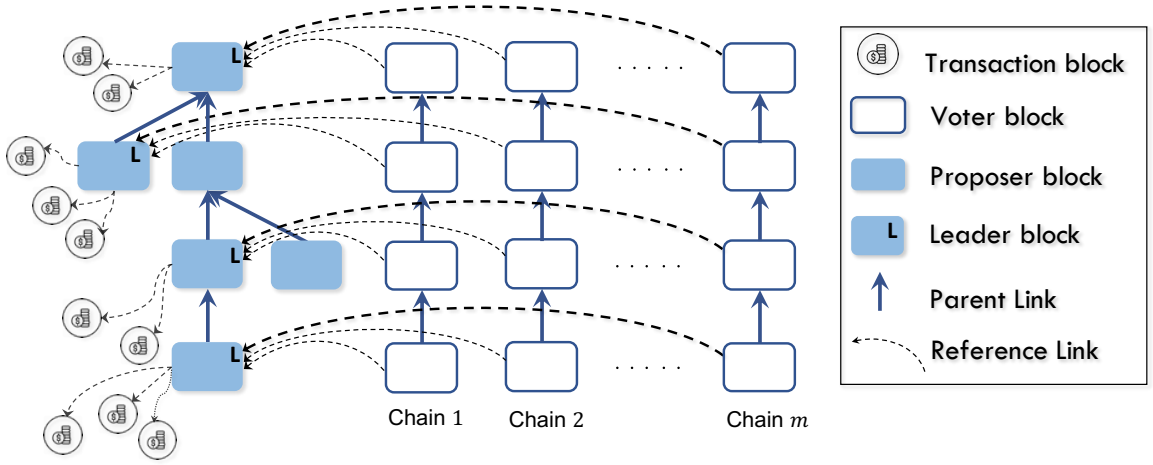

Figure 3: Factorizing the blocks into three types of blocks: proposer blocks, transaction blocks and voter blocks.

rule in the proposer tree. We define the *level* of a proposer block as its distance from the genesis proposer block, and the *height* of the proposer tree as the maximum level that contains any proposer blocks. To determine the ordering of proposer blocks (and thus transaction blocks and transactions), we elect one *leader* proposer block from each level. The sequence of leader blocks up to the height of the proposer tree is called the *leader sequence*, and is determined by the *voter* chains. Note that the leader blocks do not need to follow the chain structure of the proposer blocks because otherwise deadlock may occur if conflicting blocks (i.e., two proposer blocks not on one chain) are determined as leader blocks.

In Prism, there are m voter chains, where $m \gg 1$ is a fixed parameter chosen by the system designer. The larger the m, the more parallel the voting process and hence the shorter the latency of confirmation. In general m is chosen as large as network bandwidth and memory management issues are manageable. For example, $m = 1000$ is chosen in the full-stack implementation of Prism. New voter blocks are mined on each voter chain according to the longest chain rule. A voter block votes for a proposer block by containing a reference link to that proposer block, with the requirements that: (1) a vote is valid only if the voter block is in the longest chain of its voter tree; (2) each voter chain votes for one and only one proposer block at each level; (3) each voter block votes for all the proposer levels that have not been voted by its parent. The leader block at each level is the one that has the largest number of votes among all the proposer blocks at the same level (ties can be broken by the hash of the proposer blocks). The elected leader blocks then provide a unique ordering of the transaction blocks to form the final ledger.

Prism also uses cryptographic sortition to prevent the adversary from focusing its mining power on a specific type of blocks or on a specific voter chain. A miner first forms a "superblock" containing $m+2$ parts: a transaction block, a proposer block and a voter block on the i-th voter tree ($1 \leq i \leq m$). We say a superblock is successfully mined if

$$Hash(\mathsf{nonce}, \mathsf{superblock}) < T_{\text{tx}} + T_{\text{prop}} + mT_{\text{v}}. \tag{1}$$

Further, every successfully mined superblock is identified as a transaction block, a proposer block or a voter block based on the hash output:

- identify the superblock as a proposer block if the hash output is less than T_{prop};
- identify the superblock as a transaction block if the hash output is in the range $[T_{\text{prop}}, T_{\text{tx}} + T_{\text{prop}})$;
- identify the superblock as a voter block on the i-th voter tree ($1 \leq i \leq m$) if the hash output is in the range $[T_{\text{tx}} + T_{\text{prop}} + (i-1)T_{\text{v}}, T_{\text{tx}} + T_{\text{prop}} + iT_{\text{v}})$;

Due to the random output nature of hash functions, the probability a superblock is identified as a proposer block, a transaction block or a voter block on the i-th voter tree ($1 \leq i \leq m$) is proportional to T_{prop}, T_{tx} and T_{v}, respectively. See Figure 4 for an illustration. We set the transaction target difficulty T_{tx} easy, as in Prism 1.0; this allows for a lot of transaction blocks to be mined in parallel (up to the limits of how much the network capacity can handle). The proposer and each voter chain target difficulty (T_{prop} and T_{v}, respectively) are set high, as in Bitcoin for security; this ensures that the proposer block sequence and each voter chain grow slowly enough that forking is not an issue (i.e., the growth rate λ is much smaller than $\frac{1}{\Delta}$, the network delay).

The Prism protocol is similar to other ideas we have seen in terms of applying the decoupling principle (Prism 1.0, Fruitchains, and Bitcoin-NG all employed this principle to different degrees). However, it is different in a special way:

> The actual confirmed sequence of proposer blocks may never form a chain.

This means that the miner can no longer validate a block *before* mining because the confirmed proposer blocks may not form a chain and the miner cannot know the current state of the ledger given a

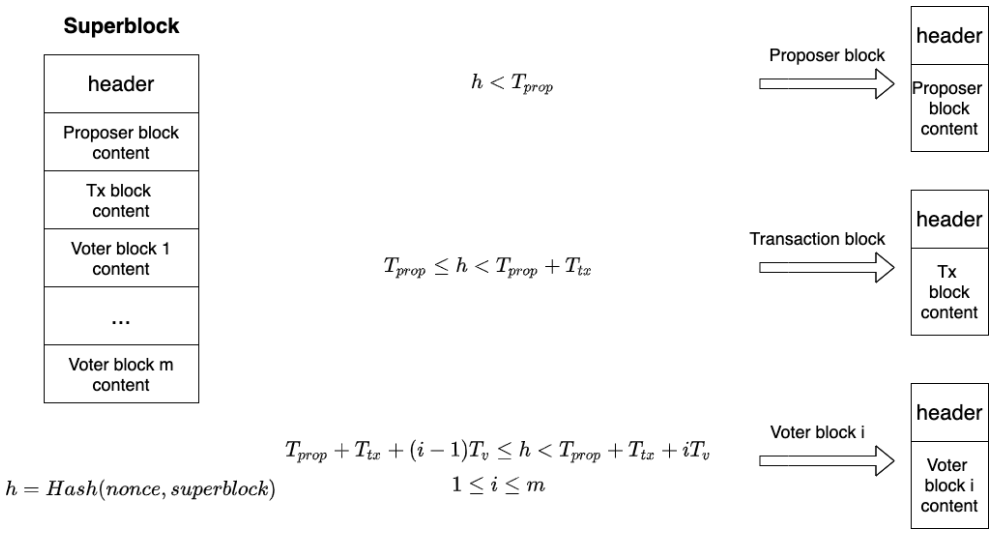

Figure 4: Superblock containing a transaction block, a proposer block and m voter blocks is "$(m+2)$-for-one" mined; the resulting cryptographic sortition identifies the type of the block as the winner of the mining process.

proposer parent block. So Prism decouples the process of mining blocks from validating transactions. Note that given a confirmed leader sequence of proposer blocks, the ordering of transaction blocks is uniquely determined. This gives us a total ordering of transactions, and enables us to determine the validity of each individual transaction by executing them following this ordering. For example, among transactions that try to spend the same coin, only the first is considered valid. This process is called ledger sanitization and is executed after the ledger has stabilized (i.e., confirmed).

Fast confirmation rule in Prism

The parallel chain structure of the voting scheme in Prism enables low confirmation latency. The high-level idea is that votes accrue only sequentially in Bitcoin, whereas in Prism, votes accrue in *parallel*. Thus, since voting blocks are created in parallel, for a fixed amount of security, confirmation can occur in a much shorter amount of time; this reduces the latency. By how much is the latency reduced and how to arrive at a principled confirmation rule that is secure and yet affords fast latency? This is discussed next.

We look at the confirmation latency of Prism in a simple case, where a single honest proposer block B_p at level one is mined at time 0. In Prism, B_p can be confirmed if no other proposer block at level one can receive more votes than it. But this is not operational, since the confirmation event depends on the future. One naive solution would be: we confirm B_p if it receives more than $m/2$ votes from the voter blocks that are at least k-deep in their voter chains. If we choose k as large as in Bitcoin, then this confirmation rule is safe because all these votes are permanent in the voter chains with high probability by the safety of Bitcoin. But this requires waiting long enough for the vote to be k-deep in each of the chains, leading to latency even worse than that of Bitcoin.

The key point is that one does not have to wait until each of the votes stabilize (i.e., are buried k-deep in their respective chains, with a large value of k). Even if each vote is ephemeral (because k is small, for instance $k = 2$ or 3) the *averaging* of all votes ensures that the overall vote is secure. This is due to the law of large numbers; a similar concept is *boosting* in machine learning where many low precision classifiers are combined to create a high precision classifier. This phenomenon is best described in the context of a simple attack, that is analogous to Nakamoto's private attack

on the Bitcoin protocol.

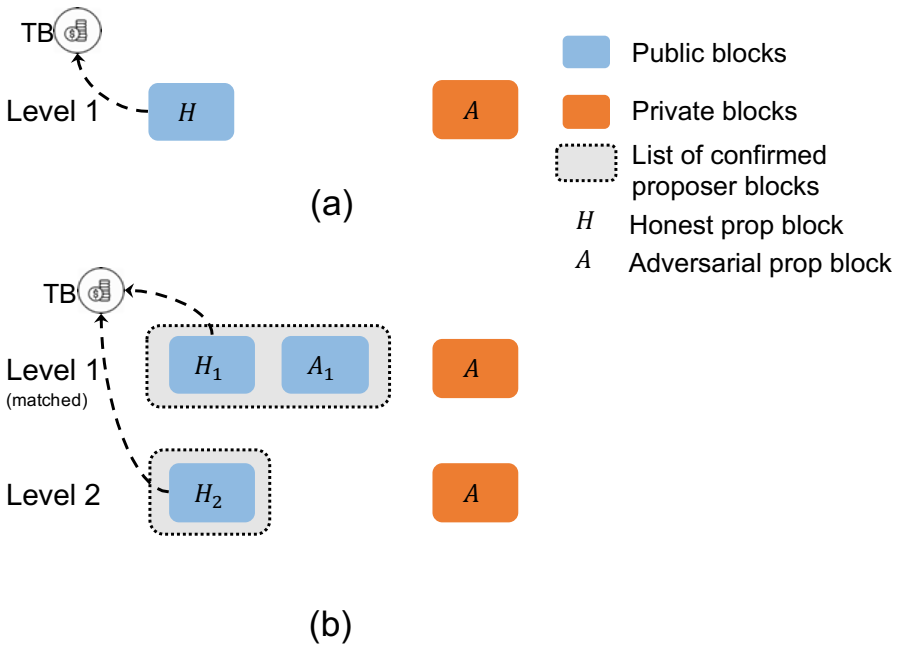

Figure 5: (a) Transaction block is referred to by an isolated honest proposer block. (b) Transaction block is referred to by a non-isolated proposer block but on the next level there is an isolated proposer block. Note that since H_2 is honest, it refers to all unconfirmed transaction blocks, i.e., TB.

Consider a situation when a transaction block TB is referred to by an honest proposer block H which is currently isolated at its level, i.e. no other public proposer block exists at the same level. See Figure 5(a). This case is quite common since the mining rate of the proposer blocks is chosen such that there is little forking in the proposer tree. Block H will start collecting votes, each of which is on the longest chain of its respective voter tree. Over time, each of these votes will become deeper in its voter chain. An attack by the adversary is to mine a private proposer block A at the same level, and on each of the voter trees fork off and mine a private alternate chain and send its vote to the block A. After leader block H is confirmed, the adversary continues to mine on each of the voter alternate chains to attempt to overtake the public longest chain and shift the vote from H to A. If the adversary can thereby get more votes on A than on H, then its attack is successful.

This attack can be viewed as the m-chain analog to Nakamoto's private attack on Bitcoin, where instead of having one race between the honest chain and the private chain we have m such races. In fact, Nakamoto's calculations on the success probability of an attack on a single chain can help us determine how deep we need to wait for the votes to become to confirm the proposer block H. At tolerable adversary power $\beta = 0.3$, the reversal probability in a single chain is 0.45 when a block is 2-deep. With $m = 1000$ voter chains and each vote being 2-deep, the expected number of chains that can be reversed by the adversary is 450. The probability that the adversary got lucky and can reverse more than half the votes, i.e. 500, is about 10^{-3}. Hence to achieve $\epsilon = 10^{-3}$, we only need to wait for 1000 votes each 2-deep. This incurs much shorter latency than the 24 block depth needed for *each* vote to be reversed with probability 10^{-3}. This reduction in latency is conceptually similar to averaging many unreliable classifiers to form a strong aggregate classifier: the more voter chains there are, the less certainty of permanence each individual vote needs to be, thereby reducing confirmation time. This gain comes without sacrificing security: each voter chain is operating slowly

enough to tolerate β adversarial hash power.

Fast confirmation rule. How to convert the intuition of fast latency of Prism to a principled confirmation rule? Note that the confirmation rule would have to be secure under all possible attacks, not just the specific private attack described above. The key idea is to confirm at a time t^* such that if there is no competing proposer block appears at level one before t^*, then no other proposer block can accumulate more than $m/2$ votes anymore after t^*, with high probability. An observation from the longest chain protocol is that a prefix of the longest chain will stabilize as the chain grows longer, i.e. as time passes (because the chain grows with time). Therefore, in Prism, since each voter chain adopts the longest chain rule, the fraction of votes that have stabilized is also monotonically increasing in time. By the law of the large numbers, when $m \to \infty$, if at a certain time each vote has stabilized with probability $1/2$, then no other proposer blocks can realistically accumulate more than $m/2$ votes anymore, so that we can confirm the block. Then we can have the following time-based confirmation rule.

Tentative Confirmation rule. We confirm B_p at time t^* if there is no competing proposer block, where t^* is the solution of the equation $h(t) = 1/2$ and $h(t)$ is a function that calculates the fraction of stabilized votes (or equivalently the effective vote from one voter chain) at time t. We derive t^* and a closed form expression for $h(t)$ in the appendix, depending only on the mining rate λ and tolerable adversary hash power fraction β. For instance, when $\beta = 0.3$, we can numerically calculate $t^* \approx 3/\lambda$, so the confirmation time is a small factor of the average inter-block arrival time.

However, time is not observable in a proof-of-work blockchain because the timestamp may not be accurate or even be faked in an adversarial block. But time can be estimated from the growth of the voter chains. Fortunately, the large number of voter chains in Prism makes this estimation accurate even for short time periods because of the law of large numbers. The total mining rate of all voter chains is $m\lambda$, so on average $m\lambda t$ voter blocks are mined within t unit of time. This conversion from time to block is accurate for large m, so finally we can have the following block-based confirmation rule that is observable.

Final confirmation rule. We confirm B_p when $mt^*\lambda$ voter blocks are mined and there is no competing proposer block. Again t^* only depends on the mining rate λ and the tolerated adversary hash power fraction β. When $\beta = 0.3$, we know that $t^* \approx 3/\lambda$ and we confirm the proposer block when $3m$ voter blocks are mined.

Non-isolated Proposer Block

Consider now the case when the transaction block `TB` is referred to by an honest proposer block H_1 which is not isolated at its level, i.e. H_1 is matched by an adversarial public proposer block A_1 (the competing proposer block could also be honest). This matching could persist for L levels until reaching a level when there is an isolated honest proposer block. See Figure 5(b) for the special case of $L = 1$. Let us separately consider the life cycle of an honest transaction vs. a double-spent one.

Honest transaction. A naive approach for confirming `TB` would be to wait until we can definitively confirm H_1 or A_1. However, this may be slow because of adversarial attacks that try to balance votes. A key insight is that for honest (non-double-spent) transactions, we do not need to know *which* of H_1 and A_1 is confirmed—only that one of them will be confirmed. This weaker form of *list confirmation* works because if A_1 eventually gets confirmed, a later honest proposer block can still refer to `TB`. To confirm an honest transaction at level i, we need two events: (1) list confirmation of all levels up to i; (2) an isolated honest proposer at level i. Once we have list-confirmed a set of proposer blocks at level i referring `TB` (e.g., either H_1 or A_1 will be the leader), we know that no other block can be the leader at that level. However, list confirmation alone is not enough for honest transaction confirmation if the transaction is not present in all ledgers. In that case, we also need to wait for an isolated honest proposer level, where the proposer block will include `TB` in the ledger. Once this isolated honest proposer level is confirmed *and* all the preceding levels are list-confirmed, we can be sure that `TB` will appear in the final ledger. The confirmation latency is thus the maximum

of two parts:

(1) List confirmation. We fast confirm that the adversary cannot produce a private block A with more votes than the votes of public blocks H_1 and A_1. The logic is similar to the case of isolated honest proposer block discussed above, viewing the situation as a race between honest nodes voting for the public blocks H_1 or A_1 and adversary voting for A. Adversarial actions (e.g., presenting H_1 to half the honest nodes and A_1 to the other half) can cause the number of votes to be evenly split between H_1 and A_1, which can slow down list confirmation, albeit not significantly.

(2) Isolated honest proposer level. In Figure 5(b), if we wait until level 2, we see an isolated public proposer block H_2 which can be fast confirmed. At this point, we know that the final leader sequence at levels $1, 2$ is either H_1, H_2 or A_1, H_2, both of which contain our honest transaction since H_2 refers to all previous unconfirmed transaction blocks.

Double-spent transaction. To confirm double-spent transactions, we need stronger conditions than those listed above: namely, instead of list confirmation, we need *unique block confirmation*, confirming which block at a proposer level will be the ultimate leader. This is achieved once list confirmation occurs *and* one of the list-confirmed blocks can be reliably declared the winner. If one of the public proposer blocks H_1 or A_1 gathers many more votes than the other block, then we can fast confirm a unique leader, even for double-spent transactions. However, other adversarial attacks (such as balancing the votes on H_1 and A_1) can cause the number of votes to be evenly split between H_1 and A_1, so we cannot fast confirm a leader block. In this case, we must wait until every vote on H_1 and A_1 stabilizes, in which case either H_1 or A_1 is confirmed and only one of the double-spent transactions is accepted. A content-dependent tie breaking rule can be used to break ties after votes are stabilized.

Scaling latency via Prism

The latency of Prism τ_{Prism} under the "normal path" (i.e., single proposer block for a level) is

$$\tau_{\text{Prism}} = O(\frac{1}{\lambda}),$$

which is independent of the error probability ε when we choose m to be large enough, and the constant hidden in the $O(\cdot)$ only depends on β. In comparison to Bitcoin, Prism achieves faster confirmation because it only needs to wait until the effective vote $h(t)$ is greater than 0.5. We can apply the law of large numbers horizontally (in space) to conclude that the majority of the voter chains have voted for a proposer block irreversibly, and hence we can confirm the block with high probability. While in Bitcoin there is only one chain, so we have to wait until $h(t)$ is $1 - \varepsilon$ to guarantee a small deconfirmation probability ε, where the law of large numbers is applied vertically (in time) as we have seen in Lecture 6. The comparison between Bitcoin and Prism latency is in Figure 6. In summary, by having many voter chains, Prism gets rid of the $\log_e(\frac{1}{\varepsilon})$ term in Bitcoin latency, i.e., decouples latency from security and achieves fast confirmation.

References

An analysis of the latency–security tradeoff covering all possible attacks by the adversary has been conducted in this paper for the longest chain protocol.

Scaling latency of the PoW longest chain protocol has been an active research topic in the last a few years, with limited success. Besides Prism, another protocol, called Ledger-combiners, also uses parallel-chains for achieving low latency. Ledger combiners run multiple instances of the longest chain protocol in parallel; the confirmation is conducted at the transaction level instead of the block level: if a transaction appears in a large fraction of all the parallel chains, then it can be confirmed with high probability. For conflict-free transactions, confirmation can be accelerated to a constant

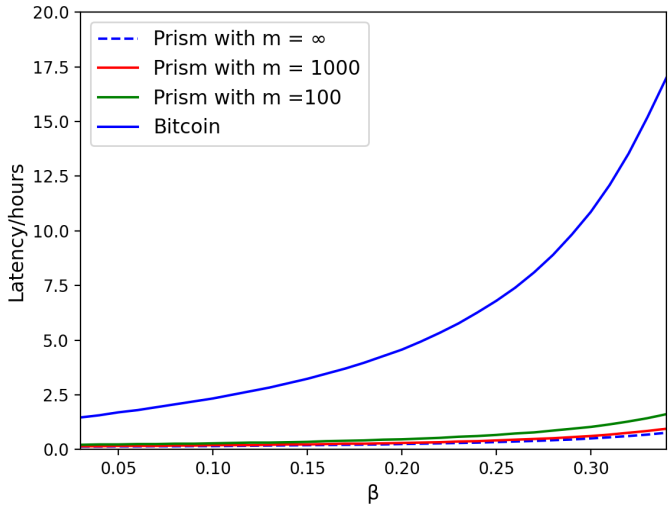

Figure 6: Comparison between Bitcoin and Prism latency under various values of β. We set λ to be 6 blocks per hour and the error probability $\epsilon = 10^{-3}$. This also justifies the choice of m in the implementation of Prism.

multiple of block time with negligible error probability (similar to Prism's guarantee on the single proposer block case).

In this lecture, we have seen that Prism's latency does not depend on the security parameter (ε), that was a dominant term in Bitcoin's latency. Achieving fast consensus among a *fixed* set of parties (also known as the *permissioned* setting) is a classical topic in computer science, studied under the rubric of called Byzantine Fault Tolerant (BFT) consensus. Recent advances have solidified the decades-old research on BFT protocols; two protocols Streamlet and Hotstuff have latency that is a small constant multiple of the worst case network delay Δ. We will study these protocols in Module 3 of this course, where we also discuss how to merge them with the permissionless setting of the PoW longest chain protocol.

Appendix

Derivation of t^*

To derive t^*, we first define the following random variables and events:

- For $1 \leq i \leq m$, let $H_i(t)$ ($A_i(t)$) be the number of honest (adversarial) blocks mined on the i-th voter tree before time t.
- Let $V_i(t)$ be the event that the i-th voter chain has voted for B_p by time t and remains voting for B_p after time t.
- Let $G_i(t)$ be the event that $H_i(\tau) > A_i(\tau)$ for any $\tau \geq t$.

Because of the cryptographic sortition used in Prism mining, $H_i(t)$ and $A_i(t)$ ($1 \leq i \leq m$) are independent Poisson random variables, $H_i(t) \sim \mathsf{Poiss}(\beta\lambda t)$ and $A_i(t) \sim \mathsf{Poiss}((1-\beta)\lambda t)$, where β is the fraction of adversarial mining power and λ is the mining rate on an individual voter chain. Let $\mathcal{V}(t) = \sum_{i=1}^{m} \mathbf{1}_{V_i(t)}$, where $\mathbf{1}$ is the indicator function, i.e., $\mathbf{1}_{V_i(t)} = 1$ if $V_i(t)$ occurs, otherwise 0.

Then we want $\mathcal{V}(t^*) \geq m/2$. However, the event $V_i(t)$ depends not only on honest and adversarial mining process, but also on the adversarial strategy that is aiming to disrupt the voter chain. So we define the good event $G_i(t)$, which is independent of the adversarial strategy. Next, we prove the following important claim:

$$G_i(t) \subseteq V_i(t).$$

To prove this claim, the logic is very similar to our analysis that the Nakamoto private attack is the worst attack. Since $H_i(t) > A_i(t)$, we know there is at least one honest block on the i-th voter chain at time t. Hence, by the voting rule in Prism, either this honest voter block or one of its ancestors should vote for B_p. And further because we have $H_i(\tau) > A_i(\tau)$ for all $\tau > t$, the voter block that votes for B_p will never be displaced from the i-th voter chain after time t, i.e., the event V_i occurs. Now when have

$$\mathcal{V}(t) = \sum_{i=1}^{m} \mathbf{1}_{V_i(t)} \geq \sum_{i=1}^{m} \mathbf{1}_{G_i(t)}, \tag{2}$$

a lower bound on $\mathcal{V}(t)$, that is independent of the adversarial strategy. Therefore, we can confirm B_p if $\sum_{i=1}^{m} \mathbf{1}_{G_i(t)} \geq m/2$. However, now the problem is that each event $G_i(t)$ is not observable at time t because the adversarial blocks may be private and also the event depends on future.

This problem can be solved by setting m to be large. Note that for $1 \leq i \leq m$, the events G_i's are independent and identical. Hence $\mathbf{1}_{G_i}$'s are i.i.d. random variables. By the law of large numbers

$$\frac{1}{m} \sum_{i=1}^{m} \mathbf{1}_{G_i(t)} \to \mathbb{E}[\mathbf{1}_{G_1(t)}] = \mathbb{P}(G_1(t)), \tag{3}$$

as $m \to \infty$.

Let $h(t) \triangleq \mathbb{P}(G_1(t))$. If we set t^* to be the solution of the equation $h(t) = 0.5$, then combing Eqn. (2) and (3), we get what we want: $\mathcal{V}(t^*) \geq m/2$. Therefore, we can think of $h(t)$ as a measure of the effective vote from one voter chain. When B_p receives a vote, there is some probability that the vote will be reversed and $h(t)$ takes this into consideration.

Now the only remaining question is how to calculate the effective vote $h(t)$. The numbers of adversarial blocks and honest blocks mined in the time interval $(0, t)$ follow Poisson distributions with mean $\beta\lambda t$ and $(1-\beta)\lambda t$, respectively. If the number of adversarial blocks is no less than the number of honest blocks at time $\tau + t$, then $G_1(t) = 0$; otherwise, from simple random walk theory, the probability that the number of adversarial blocks will ever catch up the honest from z blocks behind is given by $(\frac{\beta}{1-\beta})^z$ when $\beta < 1/2$. (Exactly the same fact has been used by Satoshi in Bitcoin white paper.) Thus, there is a closed form expression for $h(t)$:

$$h(t) = \sum_{k=0}^{\infty} e^{-(1-\beta)\lambda t} \frac{((1-\beta)\lambda t)^k}{k!} \cdot \sum_{n=0}^{k} e^{-\beta\lambda t} \frac{(\beta\lambda t)^n}{n!} \big(1 - (\frac{\beta}{1-\beta})^{k-n}\big). \tag{4}$$

which is simply a function of λ and β; see Figure 7.

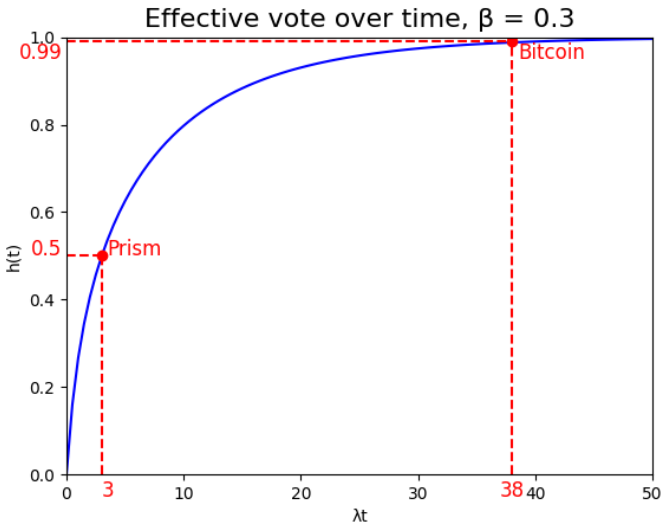

Figure 7: Effective vote $h(t)$ over time.