Lecture 6: Safety of Bitcoin

Principles of Blockchains, University of Illinois,
Professor: Pramod Viswanath
Scribes: Suryanarayana Sankagiri and Xuechao Wang

February 11, 2021

Abstract

In this lecture, we introduce a mathematical model of the Nakamoto consensus protocol, with the aim of formally analyzing the protocol's *safety*, an important security property. To this end, we introduce the common prefix property, which is closely related to the k-deep confirmation rule. We discuss different strategies that an adversary can adopt in order to violate the common prefix property. Assuming all communication is instantaneous, we see that the private attack is the worst-case attack; we see that there is a tradeoff between k and probability of error (i.e., safety of a block being violated). In particular, the error probability decays to zero as k increases, as long as honest miners control a majority of the mining power. When network delays are bounded away from zero, many more attacks are possible and thus the honest miners need to possess more than a simple majority of the mining power for the safety of Bitcoin: we characterize the exact fraction of honest mining power that is both necessary and sufficient for safety of Bitcoin (in the limit of k large). Crucially, this analysis is in the *synchronous* network setting, where all messages are delivered to every node within a fixed propagation delay. The longest chain protocol is unsafe when the network deviates from the synchronous setting.

Introduction

The last few lectures described the Bitcoin design, which is considered the prototypical blockchain system. In any blockchain system, the system's security is paramount. All aspects of the system are analyzed assuming there is an adversary that co-ordinates multiple peers so as to disrupt the system. For simplicity, we often talk of *the adversary* as a single party with considerable powers vis-a-vis other honest parties (e.g., more information, more computation power, control over the network).

There are many different security measures in Bitcoin that are designed to protect honest users from an adversary. Of these, many are simple sanity checks that honest users perform. This includes verifying the proof-of-work, verifying signed transactions and validating transactions against the current UTXO set. An adversary that tries to cheat by violating any of these can easily be identified and its messages (blocks/transactions) ignored.

In contrast, attacks at the level of the Nakamoto consensus protocol are of a different nature. Firstly, it is not always evident to honest players that an adversary is attacking the protocol. Secondly, even if players identify that the protocol is under attack, it is difficult to act against it in a consistent manner. Due to these reasons, the protocol's security analysis is more nuanced. This lecture focuses on the security of the Nakamoto consensus protocol, which is analyzed via a formal mathematical model.

The first aspect of the model is to model the mining process as a stochastic process in which new blocks appear at random intervals of time. This mathematical model serves as a basis for theoretical security guarantees. We shall see that the security guarantees are probabilistic in nature, with the randomness factor coming from the mining process.

Mining as a Poisson process

The times at which a new block is mined is modeled as a Poisson process with rate λ. Here the average inter-block time, $\frac{1}{\lambda}$, is set based on the target difficulty in the PoW mining operation; for Bitcoin $\frac{1}{\lambda} = 10$ minutes. A Poisson process is one in which new events (or arrivals) occur at random intervals following the exponential distribution. Moreover, the intervals between any two events are independent of, and statistically identical to, each other. Recall that an exponential random variable X with parameter λ has distribution

$$\mathbb{P}(X \geq t) = \exp(-\lambda t) \ \forall t \geq 0.$$

Also recall that a Poisson random variable Y with parameter λ has distribution

$$\mathbb{P}(Y = k) = \exp(-\lambda)\frac{\lambda^k}{k!} \ \forall k \geq 0.$$

In a Poisson process, the number of events in an interval of length T is a Poisson random variable with parameter λT. Moreover, the number of events in disjoint intervals of time are independent. If we consider small intervals ($\lambda T \ll 1$), there is one event in the interval with probability λT and none otherwise. Thus, a Poisson process can be emulated by counting the occurrence of heads in a sequence of (independent) coin tosses, with the probability of heads being very small. We now see why the mining process has such a property.

Imagine that the number of miners in the system and the total computing power at their disposal is constant over some period of time. Assume that each of the miners' computers are mining incessantly, and that is the only computation they are performing. It is then reasonable to say that the total number of hashes being computed (equivalently, the total number of nonces being tried out) per unit time is roughly constant. Say, a billion hashes are computed every second.

Further, suppose that the difficulty level of the hash puzzle for proposing a block is very high. E.g., say that the first thirty-five zeros must be zero for a block to be valid. The probability that a particular nonce will meet this criterion is $2^{-35} \approx 3 \times 10^{-11}$. Thus, the probability that the hash puzzle will be solved by *any miner* in a given second is 0.03, a small number (we assumed that a billion hashes are computed every second). Whether or not a proof-of-work hash is found in a particular second has no bearing on whether one will be found in the next second. The reason for this is that the hash function is essentially a random oracle; the hash values for different inputs are independent of each other. Thus, the mining process is well modeled as a Poisson process.

In modeling the mining process as a Poisson process, we focus only at the times at which new, valid blocks are created. The Poisson process model holds irrespective of the number of miners, their individual computation power, whether different miners are working on the same block or different blocks, and when different users receive newly mined blocks. The parameter λ of the mining process, called the **mining rate**, is equal to the average number of blocks mined per unit time. In Bitcoin, λ is 1/(600s), i.e., one block every 600 seconds (ten minutes). In Ethereum (which also has the same Nakamoto consensus protocol), the rate is much faster: λ is 1/(13s).

With variations in total computing power, the fixed mining rate assumption does not hold precisely. In reality, the total computation power does not change suddenly; this is especially true for a mature system like Bitcoin, which already has many miners actively participating. Thus, over a small period of time, the rate is roughly constant. Adjusting the difficulty parameter at regular intervals helps keep the mining rate at the same level. Therefore, for this lecture, we shall assume that the mining rate of the protocol is a constant λ throughout. More nuanced models with variable mining rates exist in the literature.

We now proceed to specify some other important modeling assumptions about the protocol. Some aspects of the model may deviate from reality by giving the adversary some extra power. By doing so, we are guaranteed that the statements on the system's security are guaranteed to hold even in practice.

Nakamoto consensus protocol model

We assume that there is a single adversary and many different honest parties participating in the protocol. The adversary's computing power is a fraction β of the total computing power of all users in the system. Among honest users, the computing power is divided roughly equally with each user controlling a very small fraction. This implies that typically, consecutive honest blocks are mined by different users. Such a model gives the adversary more power than a setting with a smaller number of honest users with considerable mining power. This will be made clear when we discuss the effect of network delay. Let the fraction of "hash power" of the adversary be β, and assume that $\beta < 1/2$. This means that the adversary mines blocks as a Poisson process of rate $\beta\lambda$, while honest users mine blocks at a rate of $(1-\beta)\lambda$. Further, these processes are independent of each other.

In reality, parties store all blocks that they hear of, including blocks that fork away from the current longest-chain. Their *ledger* consists of blocks in the longest chain. For the sake of modeling, assume that each honest player only stores a single blockchain at all times–the longest chain that they have heard until that time. Let $\mathcal{C}_i^h$ denote the chain held by party h at time $i \in \mathbb{R}^+$.

If an honest party hears of multiple chains with the same (maximum) length, we assume that they choose one of them arbitrarily as $\mathcal{C}_i^h$. This choice is made by the adversary. Note that this implies an honest user may swap its chain when it hears of an equally long chain, not just a strictly longer one. It also implies that if two honest parties both hear of two chains of equal (maximum) length, then the two parties may adopt different chains. This tie breaking power is another example of giving the adversary extra powers than what may exist in reality.

The *prefix* of a chain is a sub-chain consisting of the first few blocks. More formally, we say that chain $\mathcal{C}_1$ is a prefix of chain $\mathcal{C}_2$ if all blocks in $\mathcal{C}_1$ are also present in $\mathcal{C}_2$. (By default, we assume that a chain is a sequence of blocks from the genesis down to any other block.) We denote this by $\mathcal{C}_1 \preceq \mathcal{C}_2$. We define the notation of the prefix of a chain as follow.

- For a chain $\mathcal{C}$, let $\mathcal{C}^{\lfloor k}$ be the prefix chain obtained by dropping the last k blocks. In case $\mathcal{C}$ has less than or equal to k blocks, let $\mathcal{C}^{\lfloor k}$ be the genesis block.

Recall the k-deep confirmation rule in Bitcoin: a node treats all but the last k blocks in its longest chain as *confirmed*. In our notation, the blocks in $\mathcal{C}_i^{h\lfloor k}$ are confirmed by user h at time i. Once we confirm a block, we also confirm all the transactions in it. Note that the confirmation rule is *locally* applied by each user; therefore, a transaction confirmed by one user needn't be confirmed by another. However, it is desirable that a transaction confirmed by one user is soon confirmed by all other users, and remains confirmed forever after. In blockchains, this desirable property is called a *safety property*.

Formal definitions of safety

We now formally define safety of the Nakamoto consensus protocol. We give two definitions: the first is a statement about the entire duration of the protocol (a global/extensive statement), while the second concerns just one particular block (a local/intensive statement).

Definition 1 (Common Prefix Property)**.** *For a blockchain protocol, the k-common prefix property holds during an execution of the protocol if any block that is committed by one honest user appears in every honest user's chain thereafter. Mathematically, for all pairs of times $i_1 \leq i_2$, for all pairs of honest users h_1, h_2,*

$$\mathcal{C}_1^{\lfloor k} \preceq \mathcal{C}_2$$

where $\mathcal{C}_1 \equiv \mathcal{C}_{i_1}^{h_1}$, $\mathcal{C}_2 \equiv \mathcal{C}_{i_2}^{h_2}$.

Definition 2 (Individual block safety)**.** *In an execution, a block B present in some honest user's chain is safe if, after it has been committed by any honest user, it remains a part of all honest user's*

chains. Mathematically, if block B is committed by some user h at time t, then for all $t' \geq t$, for all honest users h', $b \in \mathcal{C}_{t'}^{h'}$

Note that the two statements are very similar in form. In fact, the former subsumes the latter. For simplicity of exposition, we focus on the latter property alone.

The above statements are merely definitions of a desired security property. Whether or not such a property holds (or with what probability it holds) requires further calculations. A security guarantee/theorem would be of the form that the k-common prefix property holds with probability approaching one, as k approaches infinity. Such security guarantees are given under some assumptions: assuming a fixed mining rate, a bound on the adversary's computing power, and a bound on the network delay. Crucially, these guarantees are given assuming the adversary can act arbitrarily; in other words, we do not assume anything about the adversary's strategy at all. To get a sense of how the adversary can disrupt safety, we take a look at some possible adversarial actions. In particular, we will focus on an adversary trying to disrupt the individual safety of a fixed block B.

Private attack

Consider a simple model of the blockchain system with all users split into two groups: many honest users and a singe adversarial user. The adversarial user is trying to re-write the last k blocks in the ledger by performing a private attack. Assume that the total mining rate is fixed at λ. We suppose that the adversarial fraction of hash power is β (for "bad"); the honest fraction of hash power is $1 - \beta$.

Suppose the adversary wishes to violate the safety of a block B. To do so, the adversary must ensure that block B is first confirmed by some (or all) honest users, and then, at some time in the future, must dislodge B from the longest chain, i.e., it must create a fork from a block preceding B, and must eventually produce a chain of length equal to or longer than the longest chain containing B, after B has been confirmed.

One possible attack is the private attack, introduced in Lecture 3, which we investigate here in more detail. Let B' be the parent of block B. One option is for the adversary to mine a conflicting block on B' immediately after block B' is mined. It keeps mining in private, creating an ever-increasing chain. The honest users, unaware of the private chain, continue to mine following the longest chain rule, below B. In the private attack, the adversary does not contribute to the chain the honest nodes are mining on. Note that the honest and adversarial chains are independent of each other and distributed as Poisson processes with rate $(1 - \beta)\lambda$ and $\beta\lambda$, respectively.

When should the adversary reveal its chain to the honest users? Suppose it reveals its private chain while it is shorter than the longest honest chain. The honest users will simply ignore the adversary's chain, and it will not produce any effect. Thus, the adversary must reveal its chain only when it is at least as long as the honest chain. What happens if the adversary reveals its chain too early, i.e., before block B gets k-deep? It would still end up displacing B, but then it's actions do not lead to a safety violation! Thus, the adversary must also wait until the honest chain is long enough (see Figure 1).

Analysis of the private attack under zero delay

Suppose now that the network delay $\Delta = 0$. So any node mined successfully by an honest node reaches all other nodes instantaneously. One immediate effect of this is that the honest nodes always have access to the latest mined block and thus can always mine on the tip of the longest public blockchain (by potentially switching their mining upon receiving a new block). Denote the time intervals between the honest and adversarial blocks in the chains of length $k + 1$, following block B' by $(T_0^h, T_1^h, \ldots T_k^h)$ and $(T_0^a, T_1^a, \ldots T_k^a)$ respectively (see Figure 1, where $k = 5$). Then the

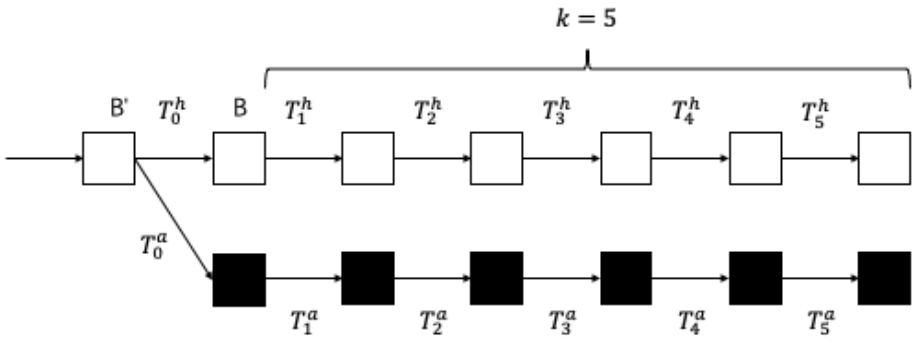

Figure 1: Private attack on block B with $k = 5$.

private attack is successful exactly when the total time to mine $k+1$ blocks by the adversary is faster than the time the honest miners take to mine their $k+1$ blocks, i.e.,

$$\sum_{i=0}^{k} T_i^h > \sum_{i=0}^{k} T_i^a.$$

Note that the times are all independent of each other and $T_0^h, \ldots, T_k^h$ are identically distributed as exponential random variables with mean $\frac{1}{(1-\beta)\lambda}$ and $T_0^a, \ldots, T_k^a$ are identically distributed as exponential random variables with mean $\frac{1}{\beta\lambda}$. By the law of large numbers,

$$\frac{1}{k+1}\sum_{i=0}^{k}(T_i^h - T_i^a) \to \frac{1}{(1-\beta)\lambda} - \frac{1}{\beta\lambda}.$$

The limiting value is positive (i.e., the private attack is successful) exactly when

$$\frac{1}{(1-\beta)\lambda} > \frac{1}{\beta\lambda},$$

or simply $\beta > \frac{1}{2}$, i.e., the adversary controls a majority of the hash power. This calculation shows that in the limit of k very large, the safety of any block B in the longest chain of Bitcoin will continue to remain in the longest chain with probability approaching one. In general there is a tradeoff between a confirmation depth of k and the resulting safety of a block B that has k blocks mined under it. The probability of "deconfirmation", i.e., the block B gets dislodged from the longest chain can be calculated as follows (here $s > 0$ is a parameter to be chosen later):

$$P(\sum_{i=0}^{k}(T_i^h - T_i^a) > 0) = P(s\sum_{i=0}^{k}(T_i^h - T_i^a) > 0) = P(e^{s\sum_{i=0}^{k}(T_i^h - T_i^a)} > 1) \quad (1)$$

$$\leq E[e^{s\sum_{i=0}^{k}(T_i^h - T_i^a)}] = E[\Pi_{i=0}^{k} e^{s(T_i^h - T_i^a)}] = \Pi_{i=0}^{k} E[e^{s(T_0^h - T_0^a)}] \quad (2)$$

$$= \left(\frac{\beta\lambda}{\beta\lambda + s}\right)^{k+1} \left(\frac{(1-\beta)\lambda}{(1-\beta)\lambda - s}\right)^{k+1}, \quad (3)$$

because of the factorization of the moment generating function of independent random variables. Choosing $s = \frac{(1-2\beta)\lambda}{2}$ minimizes the exponent and we see that the probability of deconfirmation of a block B is upper bounded by $e^{-c(k+1)}$ where the *exponent* is given by:

$$c = 2\log_e(\lambda) - \log_e(4\beta(1-\beta)\lambda^2) > 0.$$

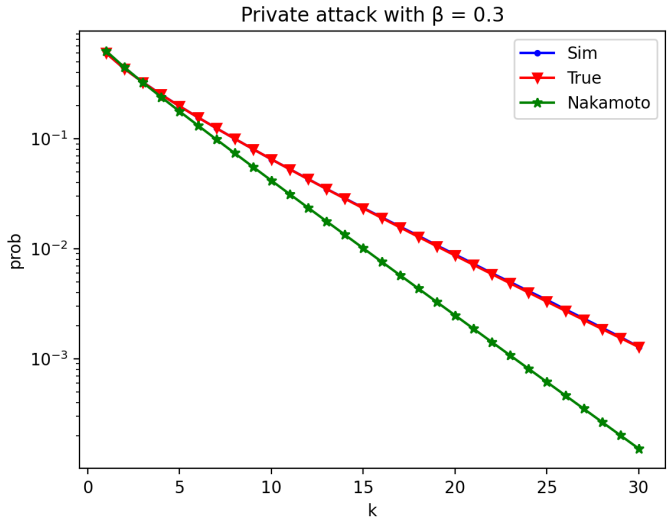

Figure 2: Private attack with $\beta = 0.3$.

So the probability of deconfirmation decays exponentially in k, as also illustrated in Figure 2. Turns out this calculation was also conducted by Nakamoto themself, albeit somewhat incorrectly, which is included in Figure 2 as a comparison.

Private attack is the worst-case attack

Private attacks, when successful, enable double-spends and constitute a particularly serious threat. However, they are not the only possible attack on the safety of the blockchain. Another canonical attack, known as the *balance* attack is the following, illustrated in Figure 3. Here the three honest blocks are denoted by B_1^h, B_2^h, B_3^h and the adversarial blocks are denoted by B_1^a, B_2^a, B_3^a. The adversary mines and releases blocks in such a way as to *balance* the heights of two chains; by such a balancing action, the adversary succeeds in switching the honest miner actions across two chains never letting any one chain stabilize. While this attack does not enable a double-spend directly, it is still a serious safety threat since the ledger keeps changing and entries never stabilize. In Figure 3, the six blocks are released in the order of $B_1^h, B_2^h, B_1^a, B_2^a, B_3^h, B_3^a$ and the longest chain switches from the left to the right and then back to the left.

The balance attack is also only an instance of the large space of adversarial actions. For instance, an attacker could launch a balance attack to *split* the honest mining power among the two chains (thus reducing the rate of growth of each one) and then launching a fatal private attack. In general, the space of adversary strategies is very vast, growing over time, and is parameterized by two quantities: where to mine the adversary blocks, and when to publish the adversary blocks. However, in the special case when the network delay is zero, it turns out that whenever any attack on safety is successful, the private attack is also successful. This is true for each instance of the when and where the honest and adversarial mining actions occur (i.e., for every *sample-path* of the underling Poisson mining processes). Suppose the attack is launched on the block just below the genesis block (left block on the first level in Figure 3; here $k = 2$). Consider two random variables A_k, H_k which represent the number of blocks mined by the adversary and the honest nodes respectively during an attack on the safety of the k-deep confirmation rule. There are two main observations.

- The total number of blocks mined is $A_k + H_k$ must be at least $2k + 2$ since to launch a safety

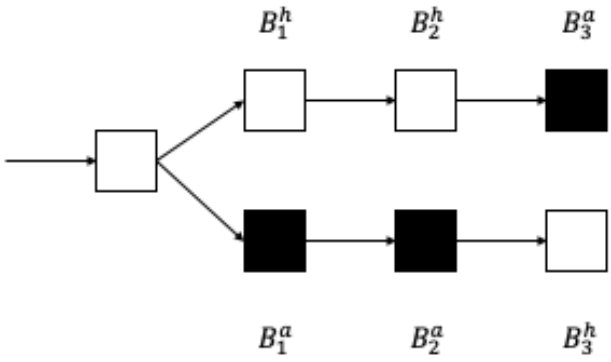

Figure 3: Illustration of a balance attack.

attack at least two chains of length $k+1$ must be present: one of the chains of length $k+1$ must have been present to confirm the block B and another chain of length $k+1$ must be present to deconfirm the block B by switching the longest chain. So $A_k + H_k \geq 2k+2$.

- We note that two honest nodes can never be mined at the same level of the blockchain; this is because the network delay is zero and miners instantaneously learn of any newly mined block and all honest nodes have full consensus on the longest chain. Since honest nodes mine on the tip of the longest chain, there could not be parallel mining by the honest nodes at the same level. So to have two chains with both length at least $k+1$, it must be that the adversarially mined blocks must be larger in number than the honest ones: $A_k \geq H_k$.

From the above two observations we conclude that $A_k \geq k+1$, i.e., the the adversary has already successfully mined at least $k+1$ blocks by the time of the safety attack on the k-deep confirmation rule. For the same sample path, the adversary could have used its blocks to launch a private attack.

Safety analysis under bounded network delay

We have supposed the network delay is zero in the analysis above, a simplification to model the practical Bitcoin setting: inter-block arrival rate is ten minutes on average, while block propagation delay in the Internet is of the order of a few seconds on average. Empirical measurements on the Bitcoin network show that blocks reach a very large fraction of participants within ten seconds. We will make the supposition that there is a finite amount of time Δ within which all blocks reach all participants definitively; this is the so-called Δ-synchronous network model. Further we will suppose that the adversary reserves the right to deliver the blocks to different participants at a time of its choosing (this includes potentially reordering blocks) as long as the Δ-synchrony condition is met.

The main impact of the propagation delay is that the most recently mined blocks take time to reach the honest nodes and thus they are mining on *stale* blocks, that are no longer the tip of the longest chain. Successful mining is now potentially wasteful, because the mining was based on a parent block that has already been extended. In other words there is *natural forking*, even without adversarial intervention. In Bitcoin the inter-block arrival time is ten minutes, so it is very unlikely that honest nodes will mine on stale blocks. But in other blockchains such as Ethereum (which also uses the same longest chain protocol) where the inter-block arrival time is fourteen seconds, natural forking is more prevalent. What fraction of honest mining power is "wasted" due to this natural

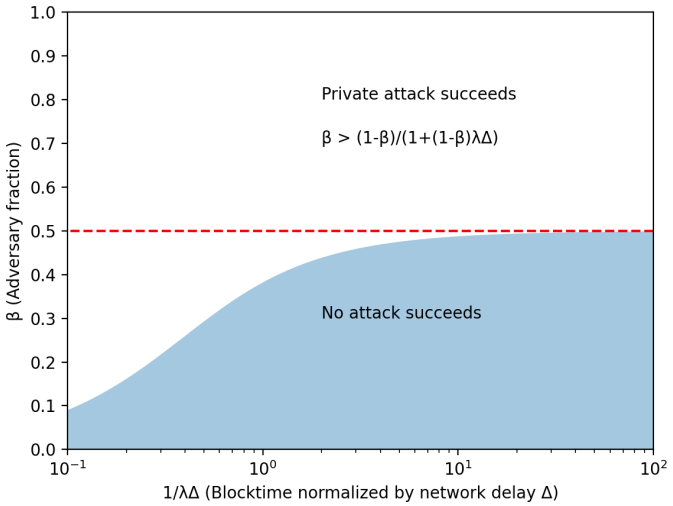

Figure 4: Minimum hash power needed by the adversary to succeed in its private attack. It turns out that this is also the security threshold for the adversary to successfully launch any safety attack.

forking? The term $(1-\beta)\lambda\Delta$ (the product of $(1-\beta)\lambda$, the rate at which honest blocks are mined and Δ, the network delay) is the expected number of blocks that are being mined "in parallel", of which only one of the blocks will succeed in extending the longest chain. Thus the rate of growth of the honest chain is now reduced by a fraction $\frac{1}{1+(1-\beta)\lambda\Delta}$ to

$$\frac{(1-\beta)\lambda}{1+(1-\beta)\lambda\Delta}.$$

In the Appendix we will mathematically justify this statement under the limit of a very large number of honest miners (each with infinitesimally small mining power). Now an adversary launching a *private attack* is successful if its rate of growth (of a private chain) is larger than that of the (reduced) growth rate of the honest chain:

$$\beta\lambda > \frac{(1-\beta)\lambda}{1+(1-\beta)\lambda\Delta}. \tag{4}$$

Figure 4 illustrates the minimum hash power adversary needs to succeed in its private attack.

As we have seen earlier, the private attack is only one of the possible strategies adopted by the adversary. When the network delay $\Delta = 0$, we have seen that the private attack is the worst-case attack in a very general setting: for every sample path of the mining process and for every confirmation depth k, the private attack is successful whenever any attack is successful. Unfortunately this is not true as seen in the example in Figure 5.

Nevertheless, it turns out that the private attack is still a worst-case attack in the sense of the minimum required honest hash power while ensuring safety against all attacks (this safety is with probability approaching one, in the limit of k approaching infinity). So the condition in Equation (4) derived for the success of the private attack also holds for *any attack*. Some intuition for why this could be true is provided next.

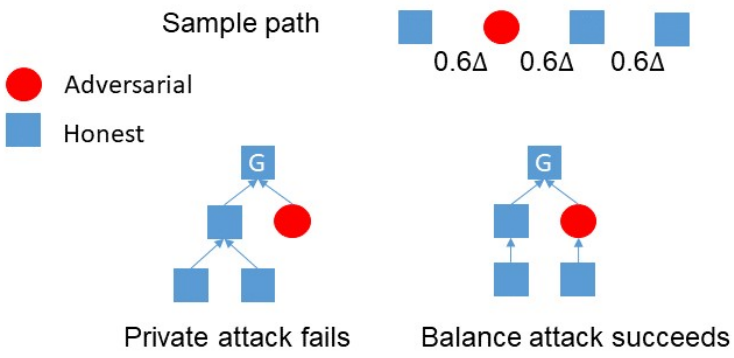

Figure 5: Attacks aimed to create a fork with length 2

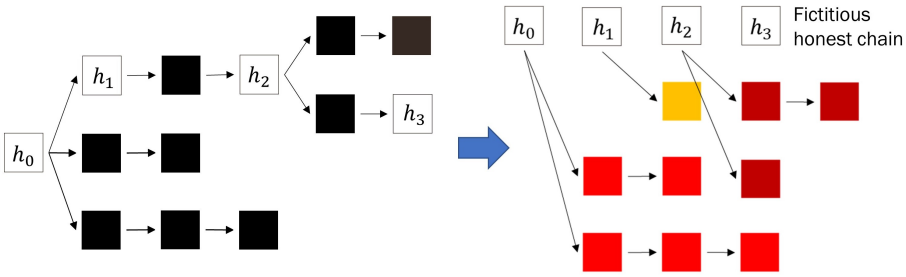

Figure 6: By blocktree partitioning, a general attack is represented as multiple adversarial chains simultaneously racing with a fictitious honest chain. Note that this fictitious chain is formed by only the honest blocks, and may not correspond to the longest chain in the actual system. However, the longest chain in the actual system must grow no slower than this fictitious chain.

Nakamoto blocks

The entire blocktree, consisting of both honest and adversarial blocks (public or private), can be arbitrary under a general attack where the adversary can make public blocks at multiple time instances. However, what we can observe is that by partitioning the complex blocktree into sub-trees, each rooted at a honest block and consisting otherwise entirely of adversarial blocks, one can view the general attack as initiating *multiple* adversarial sub-trees to race with a single fictitious chain consisting of only honest blocks (Figure 6).

The growth rate of each of these adversarial sub-trees is upper bounded by the growth rate of the adversarial chain used in the private attack. Therefore, if the private attack is unsuccessful, we know that the growth rate of each of the adversarial trees must be less than that of the fictitious honest chain. A recent paper shows that under this condition, there must exist honest blocks, which are called *Nakamoto blocks*, each having the property that *none* of the past adversarial trees can *ever* catch up after the honest chain reaches the block (Figure 7). These Nakamoto blocks serve to stabilize the blockchain: when each such block enters the blocktree, complex as it may be, it is guaranteed that the entire prefix of the longest chain up to that block remains immutable in the future[1]. When Nakamoto blocks occur and occur frequently, the safety of the protocol is guaranteed.

[1]Thus, Nakamoto blocks have a god-like permanence, they exist, but nobody knows which block is a Nakamoto block.

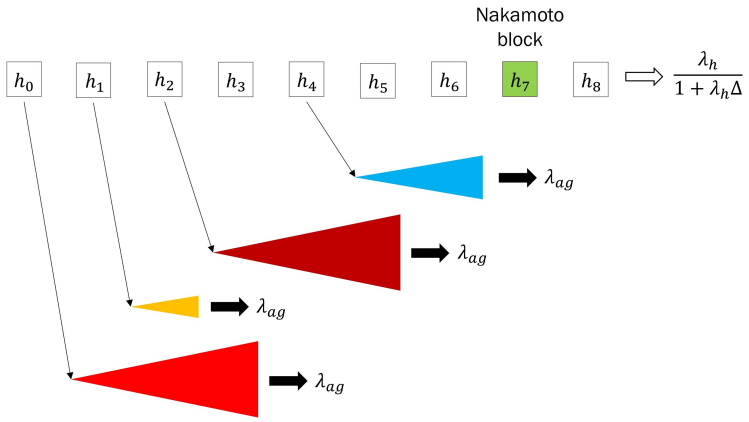

Figure 7: Race between the adversarial trees and the fictitious honest chain. While there may be multiple adversarial trees simultaneously racing with the honest chain, the growth rate of each tree is bounded by the growth rate of the adversarial chain in the private attack. An honest block is a Nakamoto block when all the previous adversarial trees never catch up with the honest chain past that block. To simplify notations, $\lambda_{ag} = \beta\lambda$ and $\lambda_h = (1-\beta)\lambda$.

Importance of synchronous network

The main point from the safety analysis of this lecture is that the longest chain protocol is safe as long as the adversarial hash power is small enough, as a function of the mining rate and worst-case network delay. It is important that the worst-case network delay be finite: in a purely asynchronous network (no guarantees whatsoever on the network delays), the adversary can mine alternate chains and share them separately with different subsets of honest nodes thus breaking consensus. It is interesting to consider a network scenario that is in-between: the network delays are finite but unknown – after an unspecified time (known as the *global stabilization time* (GST)), all messages are guaranteed to be delivered to all the nodes within bounded time. Such a model is known as the partially synchronous network model and in this case, safety can be violated because k could be smaller than GST (unlike GST, k is a finite and prespecified quantity). However, after GST the consensus returns to all the honest nodes. In later lectures we will study consensus protocols that guarantee safety even under partially synchronous network settings.

Appendix

Growth rate of honest chain under network delay Suppose that a certain honest block B is mined by a miner P at time t and B is the first block at level ℓ. In the worst case, B is not received by the remaining honest miners until time $t+\Delta$. Since the network has an very large number of honest miners (each with infinitesimal mining powers), it is very unlikely that P will mine another block before time $t+\Delta$. Since other miners have not seen B, their block will still be mined at the same level as B (with a parent block at the level $\ell-1$). We assume the honest mining process is a Poisson process with rate $(1-\beta)\lambda$, then on average $(1-\beta)\lambda\Delta$ honest blocks are mined in the time interval $[t, t+\Delta]$. These blocks will not increase the length of the longest chain as they are all at level ℓ. The first block mined after $t+\Delta$ will increase the length of the longest chain by one. Thus on average, only 1 out of $1+(1-\beta)\lambda\Delta$ blocks will increase the length of the longest chain. Thus the growth rate of the longest chain is $\frac{(1-\beta)\lambda}{1+(1-\beta)\lambda\Delta}$.

Alternatively, the inter-arrival time of the Poisson process is an exponentially random variable

with mean $1/((1-\beta)\lambda)$. Hence the average time taken to mine a block at a new level and send it to all other miners (in the worst case) is $1/((1-\beta)\lambda)+\Delta$. Only when this block is received by the other miners, can the length of the longest chain grow. Therefore, the length of the longest chain grows by one with time $1/((1-\beta)\lambda)+\Delta$ on average, which gives the growth rate $\frac{1}{1/((1-\beta)\lambda)+\Delta} = \frac{(1-\beta)\lambda}{1+(1-\beta)\lambda\Delta}$.