Poster: Low-latency blockchain consensus

Cristina Basescu
EPFL
Email: cristina.basescu@epfl.ch

Lefteris Kokoris-Kogias
EPFL
Email: eleftherios.kokoriskogias@epfl.ch

Bryan Ford
EPFL
Email: bryan.ford@epfl.ch

***Abstract*—Blockchains are increasingly attractive due to their decentralization, yet inherent limitations of high latency, in the order of minutes, and attacks on consensus cap their practicality. We introduce Blinkchain, a Byzantine consensus protocol that relies on sharding and locality-preserving techniques from distributed systems to provide a bound on consensus latency, proportional to the network delay between the buyer and the seller nodes. Blinkchain selects a random pool of validators, some of which are legitimate with high probability, even when an attacker focuses its forces to crowd out legitimate validators in a small vicinity.**

I. Introduction

While decentralized blockchains are getting increasingly attractive due to their third-party independence [2], [3], [9], they still suffer from fundamental limitations to increase throughput and lower latency [7], and to remain secure under attacks [4]. There is abundant work to fix the security and scalability problems of Bitcoin (and of blockchains that inherit some of its design decision). Existing solutions, however, still do not achieve real-time latencies, especially under heavy load, or they are insecure under strong network adversaries. These fixes fall under two categories:

- *Change the consensus algorithm* among transaction validators to practical Byzantine fault-tolerant (PBFT [5]) versions [2], [6], however, PBFT does not scale by design to large consensus groups. Thus, the system latency and bandwidth usage degrade with an increasing number of nodes. Kokoris et al. [3] try to resolve this issue by splitting the state, but the performance still remains far from real-time.
- *Scale Nakamoto consensus* by increasing the block frequency or block size [1], [8], however, the resulting systems suffer from the same security shortcomings as Bitcoin.

We believe latency should be treated as a first-class citizen in blockchains, rather than a by-product of system design. Even more so if the goal is, for example, a latency comparable to Visa, in the order of seconds. Many of the proposals described above introduce valuable directions, such as sharding, which we use in our design too. In addition, we ask ourselves the following question: *Which techniques would decrease consensus latency, while maintaining security in a Byzantine fault tolerant scenario?*

We propose Blinkchain, a latency-aware blockchain that provides bounds on consensus latency. We retain sharding as a technique to scale horizontally. Our proposal differs in the way we select validators for the blockchain in each shard. Namely, to achieve low-latency consensus among validators, we use techniques from Crux [10], which enhances scalable distributed protocols with low latency. At the same time, we maintain the security guarantee that an adversary cannot take over a particular shard by focusing its efforts (e.g., proof of work) in the vicinity of the shard.

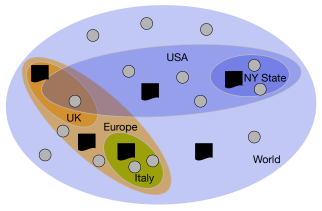

Fig. 1. Example of ring creation. The black squares represent a shard in each ring, while the grey circles represent nodes.

II. Blinkchain Architecture

Blinkchain splits the blockchain in shards on a locality-aware basis, for instance one shard per geographical area. Geographical areas can overlap, for instance a shard associated to the state of New York and a shard associated to the US. We associate to each shard a number of nearby validators, namely shard replicas, that have to reach consensus on the shard's state. As usual in Byzantine fault tolerant systems, we assume at most $1/3$ of the **total number of validators** are faulty. We explain in this section how we translate this *global property* into *shard-local guarantees* on the trustworthiness of validators.

A. Latency-aware shard creation

The goal is to ensure that transactions involving two different shards, e.g., transactions with outputs from $Shard_1$ used as inputs in $Shard_2$, incur a latency proportional to the diameter of a geographical area that covers both shards, within a polylogarithmic factor. Using the techniques in Crux [10], we create overlapping geographical areas centered around nodes in the system – the so-called *rings* – rings whose radius latency-wise increases exponentially (Figure 1). Rings incorporate as members all nodes located in that geographical area.

When a node N_a submits a transaction, it first needs to find and fetch some of its unspent transaction outputs (UTXOs) from the shard where they were created. To make UTXOs resulting from a transaction visible, validators of the transaction (Section II-C) announce the UTXOs to all outer rings that cover the shard's area. By construction, N_a is member of at least one ring that observes such announcements (the biggest ring containing all nodes in the worst case). N_a reads the announcement from the smallest such intersection ring, $Ring_i$. Thus, the total latency of the transaction between the node N_s spending the UTXOs and N_a fetching the corresponding output UTXOs is the latency to communicate in $Ring_i$ (which includes both nodes by construction), summed with the latency to verify the UTXOs, which we describe below.

The cost of the upper-bound latency guarantee is having each node participate in multiple shards at the same time, between 10 and 30 instances in Crux experiments. We believe this is an acceptable trade-off to lower latency in blockchains.

B. Preventing double spending

One challenge is to avoid double spending, specifically to prevent a node from spending the same UTXOs in two different shards. In other words, shards' views need to be consistent, such that shard validators reject double-spending transactions. For this, the system needs to trace UTXOs from their creation shard to their spending shard and validate a single trace. However, Crux only offers weak consistency between shards.

To make shards' views consistent regarding transactions, we upgrade Crux to a strongly-consistent system, as follows. We adopt a token-based approach [11], where we associate one token to each UTXOs created by a shard, token which is signed by that shard, and to which the shard appends its identity (we assume each shard forms at system setup a cothority with a collective public key, similar to ByzCoin [2]). Each UTXO used as input to a transaction carries its own token. Validators of the shard where UTXOs are spent first check the token signature of each UTXO and then refer to the origin shard, to see whether these UTXOs appear as outputs. Upon successful checks, validators insert a transaction in the origin shard to finalize token passing between the origin shard and the destination shard. Because validators always check whether the origin shard contains such a "token-spent" transaction inserted by other validators, they reject double-spending transactions.

C. Validator selection

Until now, we assumed that validators are honest in i) checking the UTXOs are valid, and ii) writing the transaction metadata to outer rings. However, if all validators of a shard are malicious, they make allow double spending, or may prevent a node from redeeming its UTXOs.

An attacker may try to exploit our locality preference for validators to create validator identities close to a shard it wants to compromise, in the hope that it crowds out other honest validators. To prove identities, we run during system initialization ByzCoin's [2] approach of using proof-of-work to solve challenges in a separate blockchain. However, a powerful attacker may solve multiple challenges, thus assume multiple identities, and send these solution to nodes conveniently close to the victim shard. If we simply assign to a shard its closest validators, an attacker only needs to be more powerful than the few other validators in the vicinity.

To ensure an attacker cannot bias validator selection in a target spending shard, we vary the boundary of the area covering the shard from where we select validators. In a first phase, we select a shard's validators using RandHound [12]. One random number selects a validator, which for a large number of validators (e.g., 600 validators for an adversary controlling 25% of the nodes) ensures with high probability that at least $2/3$ of the selected validators are honest. In a second phase, each shard asks only a subset of these validators to validate its transactions. The shard selects at verification time, based on a random beacon [12], the percentage of validators it picks from a few of its rings, e.g., 30% from $Ring_1$, 40% from $Ring_2$ and the remaining from $Ring_3$. Since the attacker does not know these percentages, it cannot know how close to the target shard to place its identities.

The security stems from the fact that the validator assignment remains random, meaning that at most $1/3$ of validators per shard are untrustworthy on a global scale. Even if the adversary tries to place all his identities close to a victim shard, the randomness guarantees that as long as there are some honest validators in $Ring_1$, $Ring_2$ or $Ring_3$, at least one of them is selected and signals any double-spending in the victim shard, before clients tentatively gain trust on transaction validity. The agreement in $Ring_2$ and $Ring_3$ can be relaxed for optimistic commitment, for instance if the total percentage of validators selected from $Ring_1$ and $Ring_2$ is higher than $1/3$ of validators, some of these validators are certainly honest.

References

[1] Bitcoin unlimited, 2016. http://www.bitcoinunlimited.info/.

[2] Eleftherios Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296. USENIX Association, 2016.

[3] Eleftherios Kokoris Kogias, P. Jovanovic, L. Gasser, N. Gailly, and B. Ford. OmniLedger: A Secure, Scale-Out, Decentralized Ledger. *IACR Cryptology ePrint Archive*, 2017:406, 2017.

[4] J. e. a. Bonneau. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *IEEE Symposium on Security and Privacy (SP)*, 2015.

[5] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI)*, 1999.

[6] C. D. et al. Bitcoin meets strong consistency. *CoRR*, abs/1412.7935, 2014.

[7] K. C. et al. On scaling decentralized blockchains (a position paper). In *3rd Workshop on Bitcoin and Blockchain Research*, 2016.

[8] I. e. a. Eyal. Bitcoin-NG: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016.

[9] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[10] M. F. Nowlan, J. Faleiro, and B. Ford. Crux: Locality-preserving distributed systems. *CoRR*, abs/1405.0637, 2014.

[11] D. Stevenson. Token-based consistency of replicated servers. In *Digest of Papers. COMPCON Spring 89. Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage*, 1989.

[12] E. Syta and P. J. et al. Scalable bias-resistant distributed randomness. In *38th IEEE Symposium on Security and Privacy*, 2017.