Using Throughput-Centric Byzantine Broadcast to Tolerate Malicious Majority in Blockchains

Ruomu Hou
National University of Singapore
houruomu@comp.nus.edu.sg

Haifeng Yu
National University of Singapore
haifeng@comp.nus.edu.sg

Prateek Saxena
National University of Singapore
prateeks@comp.nus.edu.sg

2022 IEEE Symposium on Security and Privacy (SP) | 978-1-6654-1316-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/SP46214.2022.9833617

Abstract—**Fault tolerance of a blockchain is often characterized by the fraction f of "adversarial power" that it can tolerate in the system. Despite the fast progress in blockchain designs in recent years, existing blockchain systems can still only tolerate f below 0.5. Can practically usable blockchains tolerate a malicious majority, i.e., f above 0.5?**

This work presents a positive answer to this question. We first note that the well-known impossibility of *byzantine consensus* for f above 0.5 does not carry over to blockchains. To tolerate f above 0.5, we use *byzantine broadcast*, instead of byzantine consensus, as the core of the blockchain. A major obstacle in doing so, however, is that the resulting blockchain may have extremely low throughput. To overcome this central technical challenge, we propose a novel byzantine broadcast protocol OverlayBB**, that can tolerate f above 0.5 while achieving good throughput. Using** OverlayBB **as the core, we present the design, implementation, and evaluation of a novel Proof-of-Stake blockchain called** BCube**.** BCube **can tolerate a malicious majority, while achieving practically usable transaction throughput and confirmation latency in our experiments with 10000 nodes and under $f = 0.7$. To our knowledge,** BCube **is the first blockchain that can achieve such properties.**

I. Introduction

Fault tolerance is a property of central importance in modern distributed systems such as blockchains. Fault tolerance is often characterized by the fraction f of "adversarial power" that a system can tolerate. Here the "adversarial power" may correspond to i) malicious nodes in permissioned blockchains, ii) adversarially-controlled computational power in Proof-of-Work-based permissionless blockchains, or iii) adversarially-controlled stake in Proof-of-Stake-based permissionless blockchains. Bitcoin's consensus protocol, invented over a decade ago, can tolerate f below $\frac{1}{2}$. While subsequent blockchain systems (e.g., [1], [3], [7], [11], [16], [19], [21], [33], [35]) have achieved significantly better performance than Bitcoin, all of them still can only tolerate f below $\frac{1}{2}$, or sometimes even lower. This is regardless of whether these designs are for permissioned systems, or for permissionless systems using Proof-of-Work/Proof-of-Stake.

There are growing desires, however, for blockchains to tolerate $f \geq \frac{1}{2}$. For example, there have been double-spending attacks on public blockchains, where malicious actors temporarily control more than half of the adversarial power in the network [12], [24], [27], [28]. Similarly, it has been highlighted that a few centralized miners control more than half of the power in many Proof-of-Work and Proof-of-Stake blockchains [20].

One reason why blockchains today cannot tolerate $f \geq \frac{1}{2}$ is that they often build upon *byzantine consensus* [22]. Byzantine consensus is a *one-shot* game — among other things, it requires that if honest nodes all have the same proposal for the next block, then they must all decide on that block, instead of on some adversarially-chosen block. Such requirement[1] makes it impossible to tolerate $f \geq \frac{1}{2}$. A blockchain, on the other hand, is a continuous process where usually each block in the distributed ledger is proposed by a random proposer. It is acceptable for the block to be adversarially-chosen, if the proposer happens to be malicious. Hence impossibility results on byzantine consensus under $f \geq \frac{1}{2}$ do not necessarily carry over to blockchains.

Byzantine broadcast. In our pursuit of blockchains that can tolerate $f \geq \frac{1}{2}$, we have revisited various classical primitives. We eventually focus on one such primitive — *byzantine broadcast* [6], [8], [9], [14], [25], [31], [32]. In byzantine broadcast, there is a single publicly known *broadcaster* that broadcasts an *object* (e.g., a block in blockchain) to all nodes. Some of the nodes, including the broadcaster, may be malicious. A byzantine broadcast protocol guarantees:

- All honest (i.e., non-malicious) nodes eventually output the same object (i.e., *agreement*). This object is allowed to be a special null object.
- If the broadcaster is honest, all honest nodes must output the object broadcast by the broadcaster.

Byzantine broadcast is closely related to byzantine consensus, but crucially differs from it — in particular, byzantine broadcast is solvable in synchronous systems for all $f < 1$. Starting from now on, *this paper will only be concerned with byzantine broadcast protocols that can tolerate $f \geq \frac{1}{2}$.*

While rarely mentioned in literature, byzantine broadcast can be used [26] to build blockchains. In particular, if we use a byzantine broadcast protocol that can tolerate $f \geq \frac{1}{2}$, then the resulting blockchain will immediately be able to tolerate $f \geq \frac{1}{2}$ as well. But at the same time, the resulting blockchain's throughput will also inherit from the throughput of the byzantine broadcast protocol. This turns out to be the major obstacle, because existing byzantine broadcast protocols [6], [8], [9], [14], [25], [31], [32] seriously fall short of providing acceptable throughput, as shown next.

[1] A similar requirement has led to the recent impossibility result in [10].

Throughput of existing byzantine broadcast protocols. Let us clearly define throughput. Imagine that each node has $\mathbb{B}$ available bandwidth, as provided by the deployment environment. Under such a constraint, let x be the total number of bits that a byzantine broadcast protocol can broadcast within a time period y. We define the protocol's *throughput* to be $\mathbb{T} = x/y$, and define the *normalized throughput* to be $\mathbb{R} = \mathbb{T}/\mathbb{B}$. We also call $\mathbb{R}$ as the *throughput-to-bandwidth ratio* or *TTB ratio*. This ratio essentially serves to isolate the inherent merit of the protocol from the goodness of the deployment environment, since the throughput achievable by a protocol naturally increases when deployed in a better environment offering higher $\mathbb{B}$. Obviously, $\mathbb{R}$ is always between 0 and 1, and larger is better.

Existing byzantine broadcast protocols unfortunately have rather low TTB ratios. This is perhaps not surprising, since throughput or TTB ratio has not been explicitly considered in prior research on byzantine broadcast.[2] For example, the Dolev-Strong protocol [8] and its variant [31] have $\mathbb{R} < \frac{1}{wfn}$ (see analysis in Section III), where n is the total number of nodes and w is the degree of a node in the overlay network. For $n = 10000$ and $w = 40$, the protocol has $\mathbb{R} < 3.6 \times 10^{-6}$ when $f = 0.7$. If every node has 20Mbps available bandwidth, then the protocol's throughput will be less than 0.072Kbps, even if we ignore all other overheads in implementation. A blockchain built upon such a protocol will also have a throughput of less than 0.072Kbps, which is practically unusable. As another example, in our experiments, the state-of-the-art design recently proposed by Chan et al. [6] achieves a throughput of only about 0.45Kbps under 20Mbps available bandwidth. See Section III and IX for more discussions on existing protocols.

Our OVERLAYBB **protocol.** As the first contribution of this paper, we propose a novel byzantine broadcast protocol called OVERLAYBB. OVERLAYBB is particularly suitable for large-scale systems where nodes communicate via a multi-hop overlay. OVERLAYBB achieves $\mathbb{R} = \Theta(\frac{1}{w})$, by using fragmentation, proper delay/compensation in fragment propagation, and other techniques. Here w is the degree of the nodes in the overlay. For example, if the overlay is a random graph, then w can be just $O(\log n)$. This $\Theta(\frac{1}{w})$ TTB ratio is significantly better than existing protocols. (We show later that OVERLAYBB achieves a throughput of roughly 163Kbps under 20Mbps available bandwidth.)

From byzantine broadcast to blockchain. As the second contribution of this paper, we present the design, implementation, and evaluation of a novel Proof-of-Stake blockchain called BCUBE (i.e., Byzantine Broadcast-based Blockchain, or B^3). Using OVERLAYBB as its core, BCUBE can tolerate $f \geq \frac{1}{2}$, while achieving *practically usable* transaction throughput and confirmation latency. Specifically, we have implemented a prototype of BCUBE, and evaluated its performance, with up to 10000 nodes and under similar configurations as in prior works [11], [33]. In our experiments with $f = 0.7$ and a target error probability of $\epsilon \leq 2^{-30}$, BCUBE achieves 163Kbps throughput, with one 2MB block generated about every 98 seconds, and has a transaction confirmation latency of less than 6 hours.

Such performance of BCUBE is certainly not on par with blockchains that only tolerate $f < \frac{1}{2}$. But BCUBE's throughput and latency are nevertheless *practically usable*: As a reference point, Bitcoin's throughput is about 14Kbps, with one 1MB block generated about every 600 seconds. Bitcoin entails a confirmation latency of about 9.3 hours,[3] based on the state-of-the-art analysis [17], to achieve $\epsilon \leq 2^{-30}$ under $f = 0.25$.

To our knowledge, BCUBE is the very first blockchain that can tolerate $f \geq \frac{1}{2}$, while achieving practically usable throughput and latency. There are only a few prior approaches [6], [26] for designing blockchains with $f \geq \frac{1}{2}$, which are all based on byzantine broadcast. The throughput achieved by those approaches (i.e., no more than 0.45Kbps under same setting as BCUBE) is far from practically usable. Furthermore, all those prior works are purely theoretical, ignore various practical issues, and provide no implementation.

Roadmap. The next section defines our system/attack model. Section III provides some background. Section IV and V describe the design of OVERLAYBB. Section VI presents the design of BCUBE. Section VII gives the security analysis of OVERLAYBB and BCUBE. Section VIII presents the implementation and evaluation of BCUBE.

II. SYSTEM MODEL AND ATTACK MODEL

We model hash functions as random oracles. We assume that some initial trusted setup provides a *genesis block*, which contains an unbiased random beacon to be used in the very first epoch. This is a typical assumption in Proof-of-Stake blockchains (e.g., [7], [11]).

Nodes and stakes/coins. We consider a *permissionless* setting (i.e., similar to Algorand [11]), without PKI or initial trusted setup for binding nodes to identities. Each node in the system holds a locally-generated public-private key pair, and the public key is viewed as the node's *id*. Each *node* can either be *honest* or *malicious*. The malicious nodes are fully *byzantine*, and may deviate arbitrarily from the protocol. They may also arbitrarily collude, and we view them as all being controlled by the *adversary*. We allow the adversary to be *mildly-adaptive* [7], [16], [35] — for example, it takes multiple epochs for the adversary to adaptively corrupt a node.

We rely on *Proof-of-Stake (PoS)* [7], [11], [16] for Sybil defense in our permissionless setting: We assume that there are

[2]Prior works have considered *communication complexity* (CC), which is the total number of bits sent by all honest nodes. But CC does not map to throughput or TTB ratio. For example, some protocols [6], [8], [25] require a node to send many bits in a few "busy" rounds, and nothing in other rounds. (For a given node, the adversary decides which rounds are "busy" for that node.) While such protocols may have low CC, the deployment environment still needs to provision for the high bandwidth need of those "busy" rounds.

[3]Based on the formula from [17], to ensure that the probability of the adversary (under all possible attack strategies) reverting a block is at most 2^{-30} in Bitcoin, the block needs to be at least 56 blocks deep in the blockchain. Given Bitcoin's 10-minute inter-block time, this translates to 9.3 hours. The well-known "6 blocks deep" rule of thumb in Bitcoin, and the corresponding 1-hour confirmation latency, would only give $\epsilon \approx 0.05$ [17] under $f = 0.25$.

some *stakes* (or *coins*) in the system, where the total number of coins may change over time. At any point of time, each coin has an *owner*, which is the node holding that coin. Again, the owner may change over time. Information regarding which nodes hold which coins is stored in the blockchain itself, and is publicly known. We assume that at any point of time, at most f fraction of the stakes/coins in the system are owned by malicious nodes, where f is some constant no larger than 0.99. (Our experiments mainly consider $f = 0.7$.) Sometimes as a stepping stone, we also consider a simplified permissioned setting with exactly n nodes, where we use f to denote the fraction of malicious nodes.

To simplify periodic beacon generation, BCUBE further relies on a weak Proof-of-Work (PoW) assumption: We assume that the adversary's computational power is at most 100 times of the aggregate computational power of the honest nodes. This assumption is separate from and independent of the earlier f threshold. (If needed, this "100" value can be further increased without impacting security, but at the cost of lower performance.) Note that our assumption differs from PoW-based blockchains, whose *security* depends on the adversary having *less computational power* than the honest nodes.

Communication. We assume that all the honest nodes form a connected overlay network — this is a typical assumption in large-scale blockchain systems (e.g., [11]). Consider any two neighboring honest nodes A and B in the overlay. It will be convenient to view the undirected edge between A and B as two directed edges in two directions. With respect to some δ_1 value, we say that the directed edge from A to B is *good* if a message sent by A can reach (with proper retries) B within δ_1 time, as long as the message is relatively small (e.g, $\leq$ 10KB). Otherwise the edge is *bad*. In general, under reasonably large δ_1 (e.g., $\delta_1 = 10$ seconds), one would expect that while some edges may occasionally be bad, most edges among the honest nodes will be good. Hence we assume the *honest subgraph* (i.e., the subgraph containing all the honest nodes and all the good edges) to be connected. We use d to denote an upper bound on the diameter of this honest subgraph.

Partitioning attacks [2], [30] can cause our assumption to be violated in general. But such attacks apply to many other existing blockchains as well (e.g., [2], [16], [21], [23], [30], [33]), *despite* that all these existing designs can only tolerate $f < \frac{1}{2}$. How to defend against such partitioning attacks is an active research topic by itself, and is beyond the scope of this paper: Possible defenses include hiding the overlay network structure [4], diversifying neighbors' profile [13], or preserving neighbors that provide fresher data [30].

We assume that nodes have loosely synchronized clocks, so that the clock readings on any two nodes do not differ by more than δ_2 (e.g., $\delta_2 = 2$ second). We will describe a byzantine broadcast execution as a sequence of *rounds*, and each node uses its local clock to keep track of the beginning of this execution as well as the current round number. We allow the starting time of each round on different nodes to be somewhat misaligned due to the δ_2 clock error. Each round has a fixed duration $\delta = \delta_1 + \delta_2$ (e.g., $\delta = 12$ seconds). We assume that CPU processing delay is negligible, as compared to δ. At the beginning of each round, a node receives messages, processes them, and then sends new messages. Since $\delta = \delta_1 + \delta_2$, a message sent in round i along a good edge is received by the beginning of round $i + 1$ on the receiver, as long as the message is relatively small (e.g, $\leq$ 10KB).

TABLE I: Key notations.

n	total number of nodes (for permissioned setting)
m	number of nodes (for permissioned setting) in the committee, or number of coins (for PoS setting) held by committee members
f	fraction of malicious nodes (for permissioned setting), or fraction of coins (for PoS setting) held by malicious nodes
d	upper bound on diameter of subgraph containing honest nodes/edges
w	maximum number of neighbors (both honest neighbors and malicious neighbors) that an honest node may have
ϵ	error probability
δ	round duration
l	size of object to be broadcast in byzantine broadcast protocol
s	total number of fragments (of the object to be broadcast)

Problem definition. We aim to design a blockchain system where each node maintains an append-only sequence of blocks. (BCUBE has no forks, and all blocks in this sequence are considered as "confirmed".) Each block may contain, for example, a list of transactions. The blockchain should achieve standard *safety* and *liveness* guarantees, despite the byzantine behavior of all the malicious nodes. Roughly speaking, *safety* means that the sequences on all honest nodes are consistent with each other, while *liveness* means that the sequence on each honest node keeps growing over time. We defer the exact definitions to Section VII.

Notations. Table I summarizes the notations so far, and also defines several other notations.

III. BACKGROUND ON BYZANTINE BROADCAST

We review two existing byzantine broadcast protocols [6], [8], which OVERLAYBB builds upon. To help understanding, we describe them in a simple permissioned setting with n nodes, out of which fn are malicious. We will first assume a clique topology among the n nodes, and then generalize to arbitrary multi-hop topology.

A. Dolev-Strong Protocol [8]

Clique topology. In round 0 of this protocol, the broadcaster sends the object to all nodes, with its own signature attached. Upon receiving an object in round t, if the object has less than t signatures attached (including the broadcaster's signature), a node drops the object. Otherwise the node *accepts* this object, and then adds its own signature to the object and forwards the object to all other nodes. Once an object is accepted, a node will not forward the object again in the future. A node may accept more than one object, when the broadcaster is malicious. At the end of round $fn + 1$, a node *outputs* the special null object $\perp$, if it has accepted more than one object (implying a *conflict*) or if it has accepted none. Otherwise it *outputs* the (single) object accepted.

The key intuition in this protocol is the following: *When a node B is about to forward/send an object, node B can safely accept the object if B knows that its send will cause all other honest nodes to accept this object (if they have not already done so).* This ensures that either all or none honest nodes accept that object. Specifically in this protocol, if B receives and then immediately forwards an object in round $t \le fn$, then B is sure that all other honest nodes must receive and accept this object in round $t+1 \le fn+1$, which is before the end of the protocol. On the other hand, if B sends an object in round $fn+1$, then other nodes will not receive the object before the end of the execution. But in such a case, B must have seen at least $fn+1$ signatures on the object. One of these must be from some honest node A, and A must have previously already forwarded the object to all honest nodes. Namely, A has already done the job for B.

The protocol comes with a further optimization: Once a node has accepted two objects, it no longer accepts/forwards more objects. Hence a node only sends at most two messages throughout the execution. Agreement is still preserved: If one honest node A accepts two objects, then another honest node B must also accept two objects (which may be different from what A accepts). Hence all honest nodes will output $\perp$.

Multi-hop topology. The protocol naturally generalizes [8], [31] to multi-hop topologies. The only modifications needed are: i) the protocol now runs for $fn+d$ rounds, and ii) a node now only forwards an object to its (up to w) neighbors.

TTB ratio. Consider any honest node A. In the above protocol, there are some rounds (potentially chosen by the adversary) during which A needs to forward objects to all its neighbors. Recall from Table I that l is the object size. Hence in each of those rounds, A needs to send at least lw bits total. Under the given bandwidth constraint $\mathbb{B}$, each node has the capacity to send at most $\mathbb{B}\delta$ bits in each round, where δ is the round duration. Hence the maximum l the protocol can manage is $\frac{\mathbb{B}\delta}{w}$. The protocol has total $fn+d$ rounds. Since the protocol manages to broadcast an object of size $l = \frac{\mathbb{B}\delta}{w}$ using total $(fn+d)\delta$ time, we have $\mathbb{T} = \frac{l}{(fn+d)\delta}$ and $\mathbb{R} = \mathbb{T}/\mathbb{B} = \frac{l}{(fn+d)\delta\mathbb{B}} \le \frac{\mathbb{B}\delta/w}{(fn+d)\delta\mathbb{B}} < \frac{1}{wfn}$.

B. Chan et al.'s Protocol [6]

Clique topology. Recently, Chan et al. [6][4] have proposed an elegant design to substantially reduce the number of rounds in the Dolev-Strong protocol. Chan et al.'s protocol first selects a random committee of m nodes. Now the m committee members can do byzantine broadcast among themselves, using the Dolev-Strong protocol [8] while taking at most m rounds. But it is not immediately clear how the remaining non-committee members can decide. In particular, since a majority of the committee members can be malicious, voting will not work.

[4]Chan et al.'s original protocol [6] allows a fully-adaptive adversary, but can only broadcast messages containing a single-bit. The version we describe here is for mildly-adaptive adversaries and can broadcast multi-bit messages.

Chan et al. [6] overcomes this problem in the following way. Consider one round in the Dolev-Strong protocol, where one committee member A sends a message (containing the object and signatures) to all other committee members. Their idea [6] is to replace this round with two rounds, so that A sends the message to all the non-committee members first, and then the non-committee members forward A's message (unchanged) to all the committee members. (Hence there will be total $2m$ rounds.) This enables the non-committee members to observe the communication originated from the honest committee members. Before forwarding an object, a non-committee member B can precisely predict (based on the signatures on the object) whether the committee members, upon receiving this object, will accept the object. If yes, B accepts the object before forwarding it.

Multi-hop topology. Chan et al.'s protocol trivially generalizes to a multi-hop topology, *assuming that each node has sufficient bandwidth to relay all messages*. Specifically, whenever a node needs to send messages to other nodes, it simply does a *multicast* (i.e., flooding) on the multi-hop topology, taking d rounds. Hence each of the $2m$ rounds in the clique setting now becomes d rounds, and there are total $2dm$ rounds.

IV. Design of OverlayBB

Byzantine broadcast protocols are often not complex in implementation, but their designs can be subtle. Because of this, this section focuses on intuitions. We do not aim to cover all possibilities, nor to rigorously argue for correctness here. Later, Section V presents the complete pseudo-code of OverlayBB, based on which Section VII provides formal proof for correctness and analysis of the TTB ratio $\mathbb{R}$. Such an end-to-end proof is the only way to ultimately verify the protocol's correctness.

This section considers a multi-hop topology, but to help understanding, we still assume the simple permissioned setting with n nodes. Section V later generalizes to the PoS setting.

A. Avoid Relaying Unlimited Number of Objects

Chan et al.'s protocol [6] serves as a starting point of our design. When used in multi-hop topologies, their protocol relies on the implicit assumption that *a node has sufficient bandwidth to relay all possible multicast messages*. This section first shows that such an assumption can prevent the protocol from guaranteeing agreement in practice, namely, when $\mathbb{B} \neq \infty$. We then propose a simple solution to fix this problem. Section IV-B and IV-C later propose more techniques to improve $\mathbb{R}$, to eventually get OverlayBB.

Chan et al.'s protocol in multi-hop overlay. In a multi-hop topology, each node in Chan et al.'s protocol simultaneously plays two roles: First, a node is either a committee member or a non-committee member. Second, a node is always a relaying node in the overlay for the purpose of multicast, and it needs to relay all multicast messages. Now a malicious broadcaster can generate many objects, all with valid signatures from itself. The malicious committee members can add further signatures to these objects, and then multicast all these objects. Since there can be unlimited number of such objects, eventually honest nodes will not have sufficient bandwidth to relay all multicast messages. Some message x hence will not be properly propagated to all nodes (in time). It is possible that none of the other objects are eventually accepted by any honest nodes, while the object in x is eventually accepted by some honest nodes. Since the propagation of x was not properly done, the object in x may not be accepted by other honest nodes, which then violates agreement/correctness.

To gain deeper insight, it helps to see why this problem does not exist when the protocol runs over a clique. In a clique, a node only plays a single role of either a committee member or a non-committee member. While a node also needs to forward messages there, *a node always first accepts an object before it forwards the object*. Throughout the execution, each node accepts at most 2 objects, and hence sends/forwards at most 2 messages. This is regardless of how many objects are injected by the malicious nodes. Now with multi-hop propagation, upon receiving a certain object x, a relaying node B *cannot tell whether x will be accepted* (despite B knowing an upper bound d on the diameter of the network).[5]

Our observation. Our solution to the above problem will be based on the following observation: When a node in the overlay network relays an object, while it cannot predict whether the object will be eventually accepted, the node can nevertheless determine how "promising" it is for the object to be accepted. Define a *push* to be the event of a node sending/forwarding/relaying a certain object x, together with y signatures on x, in a certain round t. Intuitively, smaller t and larger y make it more likely for x to be later accepted. More precisely, we assign each push a *score* of $2dy - t$ to summarize how promising it is, based on the following intuition: Roughly speaking, each signature gives the object an extra "lifespan" of $2d$ rounds, and an object will be accepted as long as it is received during its lifespan. The score $2dy - t$ is then the residual "lifespan" when the push is done in round t. We call a push with a higher score as a more *promising* push.

Now consider any node B, and all the pushes that B has ever done. Conceptually, if all the other pushes in the network are triggered either directly or indirectly by B's pushes, then we will have the following nice property: *If an object contained in a more promising push is eventually not accepted, then no objects contained in less promising pushes will ever be accepted.* Similarly, two objects contained in two pushes with the same score must have the same outcome: *They are either both accepted or both rejected.* (Our proofs later will formalize these properties, and also fully capture the interactions among pushes done by different nodes.)

Our solution. With the above observation, let us proceed with the design of OVERLAYBB. There may be many objects that a node B needs to forward in a certain round t. In our design, node B simply chooses the 2 objects whose corresponding pushes would be the most promising, and forwards those 2 objects (effectively "materializing" those 2 pushes). Tie-breaking can be done arbitrarily. Note that since t is fixed here, those will simply be the 2 objects with the most number of signatures. (We nevertheless introduced the score of $2dy - t$, to facilitate later discussion.) If needed, to save storage space, B can further discard all objects other than those 2 objects.

Some intuitions. Section VII will give formal correctness proofs, but we provide some quick intuitions here. First, forwarding 2 objects (instead of one) is necessary for correctness. For example, consider the case where there would have been 2 objects eventually accepted, if every node had materialized all possible pushes. Then forwarding only 1 object in each round would lead to a wrong result. Second, forwarding 2 objects in each round is also sufficient for correctness. Namely, not "materializing" the other less promising pushes will not cause any problem: If at least one of these two objects are not eventually accepted, then those less promising pushes would not contribute to the acceptance of any additional objects anyway. If both objects are accepted, recall from Section III-A that we no longer care about other objects, since we already have a conflict.

B. *Fragmentation, Delay, and Compensation*

Avoid forwarding in every round. The design in Section IV-A requires a node to forward 2 objects potentially in *every round*. For example, this may happen when a malicious broadcaster injects 2 objects in each round, with objects in later rounds being more promising. To further improve $\mathbb{R}$, we want to avoid forwarding objects in every round. We achieve this by using two *phases*. The first phase uses the design in Section IV-A to broadcast the hash of the object, where a node forwards up to 2 hashes in every round (regardless of how many hashes the adversary injects). At the end of the first phase, the honest nodes will all agree on a certain hash. The second phase uses the design in Section IV-A again to broadcast the object itself. We will focus on improving the second phase, since the bandwidth bottleneck will be in the second phase. Given the agreed-upon hash, in the second phase, each node now only needs to forward (once) a single object that matches the hash, in one single round. We call that single round as the "busy" round. Of course, each node may still have many signatures (for the object) to forward. But we leave that to Section IV-C.

Naive parallelism fails. We have explained that among the $2dm$ rounds in the second phase, each node has only one "busy" round. Given this, a naive attempt to improve $\mathbb{R}$ is to use simple parallelism. Namely, we break the l-size object into $2dm$ fragments, build a Merkle tree with all the fragments being the leaves, and add the Merkle proof to each fragment. The first phase will now broadcast the Merkle root. The second phase would then *conceptually* run $2dm$ parallel instances of the protocol in Section IV-A, with one instance for each

[5]The crux here is that B does not know whether x can reach all nodes in time. One naive idea is for B to refuse relaying x when the "residual lifespan" of x is less than d rounds. This does not work because other honest nodes will do so as well, which in turn means that B needs to see a "residual lifespan" of at least $2d$ rounds. This argument keeps going on without converging, from requiring $2d$ to $3d$, $4d$, and so on.

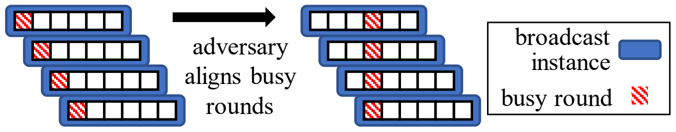


Fig. 1: Naive parallelism fails.

fragment. This seems to enable each node to fully utilize all the bandwidth in the $2dm$ rounds, with one "busy" round from each instance. Unfortunately, a malicious broadcaster controls which round will be "busy" in each instance. It can thus align all the "busy" rounds in all the instances, so that they all occur at exactly the same time (Figure 1). This defeats this naive design, regardless of how we arrange the $2dm$ instances.

Delay and compensation. Given that the adversary can choose the "busy" round for each node, we might just as well start all the parallel instances at the same time. Our first idea is that if on any node A, the "busy" rounds of two instances collide in round t, then A will send the fragment x_1 in the first instance in round t, and delay the sending of the fragment x_2 in the second instance to round $t+1$. When a neighbor B processes x_2, B should compensate, and process x_2 as if x_2 were received one round earlier. Intuitively, A is essentially telling B that because A was busy sending x_1 to B in round t, the fragment x_2 is late by one round and is only sent in round $t+1$. Since B sees that A indeed sent x_1 in round t, B should be willing to compensate.

If the overlay topology were a line topology, the above idea would work. In a more general topology, however, things get complicated. For example in Figure 2, node C_1 sends x_1 to A in round $t-1$, while C_2 sends x_1 and x_2 to A in round t and $t+1$, respectively. Then A will send x_1 to B in round t, and x_2 to B in round $t+2$. Despite all nodes being honest in this example, B sees a one-round "gap" between A's forwarding of x_1 and forwarding of x_2. Generalizing this example can make this "gap" contain many rounds. In such a case, B cannot be sure how much it should compensate — in fact, since A could be maliciously and intentionally add the "gap", B cannot even decide whether to compensate at all.

A classic result and its intuition. Before presenting our solution, we revisit a classic result [29] on competing propagations in networks. Let x_1 through x_s be the total s fragments of the object. Let us focus on the instance for x_s, while assuming for now that all other instances already work. (Section IV-C will show that the last instance is the key.) The propagation of x_s may get delayed due to competing fragments in other instances.

The classic result in [29] tells us that x_s can be delayed by at most $s-1$ rounds. In particular, this is not $d \times (s-1)$ rounds, and is regardless of how the nodes prioritize different fragments during propagation. The intuition behind this result is also important. The intuition is that if x_s is delayed at node A for y rounds, then A must have been busy sending some other y fragments. Once A forwards those y fragments before x_s, downstream nodes will have y fewer remaining opportunities to delay x_s.

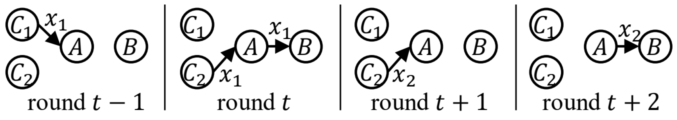


Fig. 2: B sees a gap (i.e., round $t+1$) between A's forwardings.

Our solution. Guided by the above intuition, OVERLAYBB does not have each node individually determine the amount of compensation. Instead, we use a *fixed amount of compensation* together with a *forerunner rule* during forwarding. Specifically, OVERLAYBB gives a fixed compensation of $s-1$ rounds for x_s: Whenever any node B is about to send x_s in round t, node B decides whether to accepted x_s, as if x_s were about to be sent in round $t-(s-1)$. (If $t-(s-1)<0$, we view it as 0.) For example, if B is a committee member, then B checks whether the number of signatures on x_s is at least $\frac{t-(s-1)}{2d}$.

Next, OVERLAYBB requires nodes to follow a simple *forerunner rule* during forwarding: Before a node sends x_s, it is required to have already sent all the other $s-1$ fragments. (Those $s-1$ fragments can be sent in any ordering and in any rounds, potentially with "gaps" among such forwardings.) By the earlier intuition, doing so ensures that when x_s is sent and when the compensation of $s-1$ is applied, all the possible delays for x_s have already occurred, and there will be no further delays for x_s during propagation. Note that since an honest node needs to forward all fragments anyway, the restriction from the forerunner rule has no negative effects on honest nodes. If a malicious node sends x_s to an honest node A, without having sent all the other fragments in previous rounds, then A simply ignores this message.

Quick summary. Section VII will give security analysis for the above design. As a quick summary, the classic result from [29] suggests that compensation of $s-1$ rounds will always be sufficient. The intuition behind this classic result, together with our forerunner rule, roughly suggests that if x_s has already experienced a delay of $s-1$ round by the time that a node B sends x_s, then x_s will not experience further delays in downstream honest nodes. This ultimately implies that the key invariant from the Dolev-Strong protocol [8] still holds: When a node accepts x_s, it knows that it can make all other nodes accept x_s within a $2d$ rounds.

C. Forwarding Signatures and Combining the Two Phases

Forwarding signatures. Section IV-B ignored the overhead of sending the signatures. To minimize such overhead, OVERLAYBB only uses signatures on the last fragment x_s. The other fragments do not carry signatures, and there is no notion of acceptance for each such fragment individually. The entire object (i.e., all its s fragments) is accepted iff x_s is accepted. To intuitively see why this works, note that by our forerunner rule, if x_s is accepted, then the node must have previously sent (and hence seen) all the other $s-1$ fragments. Thus if a node accepts x_s, it must be able to reconstruct the object from all the fragments. Furthermore, if all the honest nodes agree on whether x_s is accepted, they must also have agreement on whether the object is accepted. As a further optimization, since x_s is the only fragment carrying signatures, we want to make x_s as small as possible. To do so, the broadcaster simply chooses a random nonce as x_s, and the object is now split into only $s-1$ fragments.

Running the two phases in parallel. The design in Section IV-B requires two sequential phases: the first phase for the Merkle root and the second phase for the object itself. To further improve performance, we next explain how to run these two phases in parallel, using the following two modifications.

First, a node in the second phase needs to determine whether a fragment is a leave of the Merkle tree with root r, where r is agreed upon at the end of the first phase. When the two phases run in parallel, such determination cannot be easily made anymore. But recall from Section IV-A that every node A assigns a score to every push that it has ever done. Such a score captures how promising the push is. Now with the two phases running concurrently, in the (concurrent) second phase, node A simply uses the Merkle root r^A contained in its most promising push done so far in the first phase, as its current *guess* for r. Our later proof will show that using such a guess suffices to ensure the correctness of the protocol.

Second, let t^A be the round during which A is about to send the last fragment x_s in the second phase. Previously in Section IV-B, A would decide whether to accept x_s based on the value of t^A. Now that the two phases run in parallel, we need to adjust this part as well. Specifically, let t^A_{root} be the round during which A accepts r in the first phase, and define $t^A_{\text{frag}} = \max(t^A, t^A_{\text{root}}+s-1)$. When deciding whether to accept x_s, node A will make the decision as if x_s were sent in round t^A_{frag} (instead of in round t^A). The exact reasoning behind this t^A_{frag} term is slightly complex. For the lack of space, instead of going through a lengthy example here, we directly prove the correctness of such a design later.

V. Complete Pseudocode for OverlayBB

The PoS setting. Section IV assumed a permissioned setting. The actual OverlayBB protocol is designed for a permissionless PoS setting. With PoS, each node holds some coins (i.e., stakes). For each OverlayBB invocation, Section VI later will choose m random coins among all these coins. The nodes holding those coins then become committee members in OverlayBB. A node B may hold $x \geq 1$ chosen coins. In such a case, B's signature will be viewed as being equivalent to x signatures from x different committee members. We also call x as the *weight* of B. Among the m chosen coins, the node holding the first chosen coin will further be the broadcaster in OverlayBB. The information regarding which coins are chosen will be public — specifically, they are chosen by some random beacon, which is periodically computed and released. Hence all parties know the public keys (but not necessarily IP addresses) of all the committee members, each time before OverlayBB is invoked.[6]

Signature aggregation. OverlayBB uses signature aggregation to reduce signature size, as an optimization. One suitable signature aggregation scheme is the $\mathcal{MSP}$*-pop* scheme using BLS381, which gives aggregate signatures of size only 96 bytes [5]. The $\mathcal{MSP}$*-pop* scheme requires certain public parameters, which can easily be published in the genesis block of BCube. Each node can generate public keys independently and non-interactively, as and when needed, based on these public parameters. $\mathcal{MSP}$*-pop* additionally requires a proof-of-possession for each public key. In BCube, we simply require a node to add a transaction containing this proof to the blockchain, before it is allowed to be a committee member. In each invocation of OverlayBB, the possible signers are all the m committee members for that invocation. Hence for each aggregate signature, an m-bit vector suffices to indicate which of the m members are signers.

Consider any Merkle root x or fragment x. In our pseudocode, the set `all_sig` keeps track of all the aggregate signatures seen by a node so far. Note that `all_sig` may contain multiple aggregate signatures for x, since we do not combine multiple aggregate signatures into one. We use $\sigma(x)$ to denote the aggregate signature for x whose signers have the largest total weight, with arbitrary tie-breaking, among all aggregate signatures in `all_sig`. If there is no aggregate signature for x in `all_sig`, we define $\sigma(x) = \emptyset$. We use $|\sigma(x)|$ to denote the total weight of the signers in $\sigma(x)$. We use $\sigma(x)$.`add_my_sig()` to denote the aggregate signature obtained by adding the invoking node's signature to $\sigma(x)$. If the invoking node is already a signer in $\sigma(x)$, then $\sigma(x)$.`add_my_sig()` $= \sigma(x)$.

Algorithm 1. Algorithm 1 is the main algorithm for OverlayBB, run by every node in the system. OverlayBB has total $2dm+s$ rounds (Line 18 to 26). Here $2dm$ follows from the discussion in Section III-B, while the s rounds comes from the delay/compensation design in Section IV-B. Recall from Section II that each node uses its local clock to keep track of the beginning of the execution (not explicitly shown in the pseudo-code) as well as the progress of each round (Line 25).

The two phases, one for the Merkle root and one for the object itself, run in parallel by the design in Section IV-C. In each round, a node first adds the various received roots/fragments/signatures into the corresponding sets (Line 19 to 23). Next Line 24 invokes ForwardMerkleRoot() and ForwardFragment() to do the processing for the first and second phase, respectively.

After all these $2dm+s$ rounds, a node outputs a non-$\perp$ object iff i) the first phase has accepted exactly one Merkle root r, and ii) the second phase has accepted the last fragment corresponding to this Merkle root r.

Algorithm 2. Algorithm 2 largely follows the design in Section IV-A. In particular, Line 33 chooses two Merkle roots

[6]We will explain later that each epoch in BCube computes a beacon to select the committees in the next epoch. Hence our design allows a *mildly-adaptive* adversary as in [7], [16], [35] — namely, if it takes multiple epochs for the adversary to adaptively corrupt nodes, then the adversary will not be able to cherry-pick the committee members to corrupt, after seeing the beacon and before the committee members have done their work.

Algorithm 1 OVERLAYBB (**Parameters**: m, d, s)

```
 1: all_root ← $\emptyset$; // all received (Merkle) roots
 2: all_push ← $\emptyset$; // all pushes done so far for roots
 3: root_accepted ← $\emptyset$; // roots accepted so far
 4: $t_{\text{root}} \leftarrow \infty$; // round number when first root accepted
 5: all_frag ← $\emptyset$; // received fragments
 6: frag_accepted ← false; // last fragment has been accepted?
 7: all_sig ← $\emptyset$; // received signatures on roots and fragments
 8:
 9: if I am the broadcaster then
10:     break the object (to be broadcast) into $s-1$ fragments;
11:     pick a random nonce as the last fragment (i.e., $s$th fragment);
12:     let $r$ be the Merkle root of all these $s$ fragments;
13:     add the Merkle proof into each fragment;
14:     all_root ← all_root $\cup \{r\}$;
15:     all_frag ← all_frag $\cup$ {the $s$ fragments};
16: end if
17:
18: for $t$ from 0 to $2dm + s - 1$ (both inclusive) do
19:     receive messages from all neighbors;
20:     discard those received Merkle roots whose aggregate signa-
        tures do not contain the broascaster as a signer;
21:     add received Merkle roots to all_root;
22:     add received fragments to all_frag;
23:     add received aggregate signatures to all_sig;
24:     ForwardMerkleRoot(); ForwardFragment();
25:     wait until the current round $t$ ends;
26: end for
27:
28: if (|root_accepted| = 1) ∧ (frag_accepted = true) then
29:     return the object by combining the fragments (in all_frag)
        that correspond to the (single) Merkle root in root_accepted;
        // we will prove that there are exactly $s$ such fragments
30: else return $\perp$;
31: end if
```

Algorithm 2 ForwardMerkleRoot()

```
32: if |all_root| $\leq 1$ then top_root ← all_root;
33: else top_root ← $\{r_1, r_2\}$ such that $|\sigma(r_1)| \geq |\sigma(r_2)| \geq |\sigma(r)|$
        for all $r \in$ all_root; // tie-breaking can be done arbitrarily
34: for each $r \in$ top_root do
35:     if (I am in committee) and $(2d|\sigma(r)| \geq t)$ then
36:         all_sig ← all_sig $\cup \{\sigma(r)$.add_my_sig()$\}$ ;
37:         root_accepted ← root_accepted $\cup \{r\}$;
38:         $t_{\text{root}} \leftarrow \min(t_{\text{root}}, t)$;
39:     end if
40:     if (I am not in committee) and $(2d|\sigma(r)| \geq t + d)$ then
41:         root_accepted ← root_accepted $\cup \{r\}$;
42:         $t_{\text{root}} \leftarrow \min(t_{\text{root}}, t)$;
43:     end if
44:     send $r$ and $\sigma(r)$ to all my neighbors;
45:     let $p$ be the push corresponding to the above send;
46:     $p$.score ← $2d|\sigma(r)| - t$;
47:     all_push ← all_push $\cup \{p\}$;
48: end for
```

with the largest total weight of signers, and Line 46 computes the score of the push. At Line 35, a committee member accepts a root r if the total weight of signers is at least $\lceil \frac{t}{2d} \rceil$. This matches the intuition in Section III and IV-A, since each round in Dolev-Strong protocol [8] corresponds to 2 rounds in Chan et al.'s protocol [6] (under clique setting), which in turn map to $2d$ rounds in OverlayBB. Similarly, a non-committee member accepts a root r if the total weight of signers is at least $\lceil \frac{t+d}{2d} \rceil$

Algorithm 3 ForwardFragment()

```
49: if all_push = $\emptyset$ then return;
50: let $p \in$ all_push be the push with largest $p$.score; // tie-
        breaking can be done arbitrarily
51: let $x_1$ through $x_s$ denote the $s$ fragments corresponding to the
        Merkle root in $p$; // I may or may not have received all of them
52:
53: if (there exists any $i \in [1, s-1]$ such that $x_i \in$ all_frag and
        I have not forwarded $x_i$ before) then
54:     pick any such $i$ and send $x_i$ to all my neighbors;
55:     return;
56: end if
57:
58: if ($x_i \in$ all_frag for all $i \in [1, s]$) then
59:     $t_{\text{frag}} \leftarrow \max(t, t_{\text{root}} + s - 1)$;
60:     if (I am in committee) and $(2d|\sigma(x_s)| \geq t_{\text{frag}} - (s-1))$ then
61:         all_sig ← all_sig $\cup \{\sigma(x_s)$.add_my_sig()$\}$;
62:         frag_accepted ← true;
63:     end if
64:     if (I am not in committee) and $(2d|\sigma(x_s)| \geq t_{\text{frag}} - (s-1) + d)$
        then
65:         frag_accepted ← true;
66:     end if
67:     send $x_s$ and $\sigma(x_s)$ to all my neighbors;
68: end if
```

— this simply means that within d rounds, the root r will reach some committee member, and will be accepted by that committee member.

Algorithm 3. Algorithm 3 corresponds to (one round of) the second phase in Section IV-B. Section IV-B explained that *conceptually*, the second phase uses one instance for each fragment, with total s instances. But in each round, a node only sends message for at most one instance. Algorithm 3 chooses that instance (implicitly) at Line 54 and 58, and then processes *only* that single instance. Hence Algorithm 3 remains single-threaded, despite that it actually implements s parallel instances.

Line 50 follows the design in Section IV-C, and uses the Merkle root contained in the most promising push as a guess for the final accepted root. Line 53 to 56 follow the forerunner rule in Section IV-B. Line 59 computes t_{frag} as discussed in Section IV-C. Line 60 and 64 check whether to accept x_s, based on $|\sigma(x_s)|$, t_{frag}, and the $s-1$ compensation as discussed in Section IV-B. The actual decision rule is similar to Line 35 and Line 40.

VI. FROM BYZANTINE BROADCAST TO BLOCKCHAIN

So far we have presented our byzantine broadcast protocol, OVERLAYBB. We now explain how to use OVERLAYBB to build our blockchain, BCUBE.

Basic design. While largely neglected in the literature, blockchains can be relatively easily built from byzantine broadcast, in the following way. In a blockchain protocol, every node aims to maintain an append-only sequence of blocks, and all the sequences on all the honest nodes need to be consistent with each other. For convenience, imagine that there is a sequence of *slots*, which initially are all empty. The nodes in the system invoke OVERLAYBB periodically (e.g., every 98 seconds), and a node uses the return value from the ith invocation of OVERLAYBB as the block for the ith slot. For each invocation, the broadcaster is chosen randomly, who assembles a block and then uses OVERLAYBB to disseminate that block to all nodes. We say that a block/slot is *confirmed* if its corresponding OVERLAYBB invocation has ended. Note that since each OVERLAYBB invocation can take much longer than 98 seconds, the $(i + 1)$th invocation will start before the ith invocation ends. This effectively results in *pipelined invocations*, and at any point of time, there can be many active OVERLAYBB invocations. All these pipelined invocations can be implemented efficiently: Our actual implementation will simply use a *single thread* to loop through all the pipelined invocations, and process them one by one. We also stagger the round starting time of all these invocations, so that invocations near the end of the processing loop start their rounds a bit later.

The above basic framework is already used in Pass and Shi [26], which describes a theoretical design of a blockchain using the Dolev-Strong protocol [8] (instead of OVERLAYBB). BCUBE also follows this basic framework, but there are several practical issues we need to overcome, as following.

Choosing broadcaster/committee. We use two independent hash functions, $\texttt{hash}_1$ and $\texttt{hash}_2$, in BCUBE. The execution of BCUBE is divided into *epochs* (e.g., 1 epoch = 1 day). In each epoch $i - 1$, the nodes compute (explained later) a fresh public random beacon, denoted as $\texttt{beacon}_i$, to be used in epoch i. Recall from Section II that the *genesis block* contains an unbiased random beacon to be used in the very first epoch. Hence the genesis block bootstraps this sequential process of beacon generation.

We say that a slot/block is *in an epoch* if the starting time of the corresponding OVERLAYBB invocation is in that epoch. Note that the ending time may be in the next epoch. Let y be the last block that has been confirmed by the beginning of epoch $i-1$ (i.e., before the computation of $\texttt{beacon}_i$ starts). Let the coin distribution $\mathbb{D}$ (i.e., which nodes hold which coins) be the coin distribution immediately after block y (i.e., when we apply all the transactions in blocks 1 through y). For the kth slot in epoch i, every honest node uses $\texttt{hash}_1(k|\texttt{beacon}_i)$ as randomness to select m coins (with replacement) from $\mathbb{D}$. The holders (in $\mathbb{D}$) of these coins then become the committee for the OVERLAYBB invocation corresponding to that slot. The holder of the first coin selected will be the broadcaster. Since $\texttt{beacon}_i$, k, and $\mathbb{D}$ are all public information in epoch i, all honest nodes will select the same broadcaster/committee, if their have the same sequences of blocks prior to epoch i.

Generating beacons: Overview. Beacon generation is a central issue in PoS blockchains, and there have been a number of prior approaches [7], [11], [16]. Some of these [11], [16] do not work well under malicious majority. We build upon the approach in [7]. Roughly speaking, they [7] observe that the beacon is eventually only used to select a committee. Assuming that a random oracle is used to select the committee based on the beacon, the committee will be bad (e.g., having no honest committee member) with only exponentially small probability. Hence a computationally-bounded adversary simply will have a hard time finding a bad beacon, even if it can choose any beacon it wants.

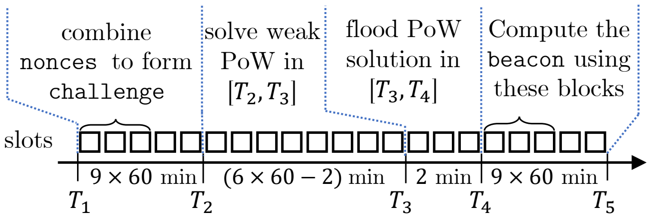

combine nonces to form challenge
solve weak PoW in $[T_2, T_3]$
flood PoW solution in $[T_3, T_4]$
Compute the beacon using these blocks
slots
9×60 min
$(6 \times 60 - 2)$ min
2 min
9×60 min
T_1
T_2
T_3
T_4
T_5

Fig. 3: Generating the beacon in an epoch. The number of slots in each portion is not to scale.

Directly adopting this idea in BCUBE does not lead to a practical solution, since the number of beacons the adversary can try is still huge. To make it work, we use a simple idea of *weak Proof-of-Work* (*weak PoW*), so that generating a valid beacon takes some computational effort. Recall that Section II assumed that the adversary's computational power is at most 100 times of the computational power of the honest nodes.

Generating beacons: Details. To facilitate beacon generation, each block in BCUBE contains two additional fields: `nonce` and `candidate`. The `nonce` field is just some uniformly random bits locally generated by the broadcaster, who is also the creator, of that block. A block is called an *honest block* if its broadcaster/creator is an honest node.

Recall that the nodes generate $\texttt{beacon}_i$ during epoch $i-1$. Let T_1 and T_5 be the start and end time, respectively, of epoch $i - 1$ (Figure 3). Let T_2 be the time when the first τ slots in epoch $i - 1$ have been confirmed. At time T_2, all the `nonce` values in these τ blocks are concatenated (not XOR-ed), and used as the fresh `challenge` for the weak PoW in this epoch. Here τ is chosen such that with high probability, there is at least one honest block (and hence one honest `nonce`) among those τ blocks. This ensures that the adversary cannot pre-compute PoW solutions before the beginning of epoch $i - 1$.

The honest nodes will try solving the weak PoW, starting from time T_2 and until time T_3, where T_3 can be any value no larger than $T_4 - d\delta$. Here T_4 is the latest time such that there are still τ slots (called *candidate-holding slots*) whose OVERLAYBB invocations have not yet started at time T_4, but will end by time T_5. To solve the weak PoW, a node needs to find x such that $\texttt{hash}_2(\texttt{challenge}|x)$ has a certain number of leading zeroes. We also call such x as a PoW solution.

At time T_3, every node will flood/multicast whatever PoW solution (if any) it has found. To avoid unnecessary bandwidth consumption, each node only sends/relays the *very first* PoW solution it finds/receives, and *ignores* all other PoW solutions. Since all the honest nodes form a connected component, if they collectively find at least one PoW solution by T_3, then every honest node B must see some PoW solution by T_4. But different honest nodes may see different PoW solutions.

The period from T_4 to T_5 serves to enable the honest nodes to agree on one PoW solution. To achieve this, from time T_4 to T_5, whenever a node is chosen as the broadcaster, it sets the `candidate` field in its block to be the single PoW solution that it previously sent/relayed, or `null` if it does not have any. Finally, at time T_5, a node examines all the τ candidate-holding slots, and picks the first block with a `candidate` value that is not `null`. It then uses $\mathtt{hash}_2(\mathtt{challenge}|\mathtt{candidate})$ as $\mathtt{beacon}_i$. Note that most likely, this `candidate` is from a malicious block and is set by the adversary. This is not a problem — all we need is that i) the adversary do not have too many candidates to choose from, and ii) the honest nodes agree on this `candidate`.

If all the τ candidate-holding slots have $\mathtt{candidate} = \mathtt{null}$, then a node will set $\mathtt{beacon}_i$ to be $\mathtt{beacon}_{i-1}$, which means that the system simply reuses the old beacon and the corresponding old coin distribution $\mathbb{D}$. Note that this can only occur when either there is no honest block in the τ candidate-holding slots (whose probability can be tuned by adjusting τ), or the honest nodes have found no PoW solution. Our analysis next will fully take into account all such possibilities.

VII. Security Analysis

This section analyses the security guarantees, or more specifically, *safety* and *liveness/throughput*, of BCube.

A. *Safety of* BCube

Recall that in BCube, each node maintains an append-only sequence of blocks. The ith block is simply the return value from the ith invocation of OverlayBB. A node invokes OverlayBB periodically (e.g., every 98 seconds), and each invocation takes the same amount of time to complete. Hence a node adds blocks, one by one, to the sequence.

Safety of BCube essentially means that for each $i \geq 1$ and after the ith invocation of OverlayBB returns, the ith block on all honest nodes should always be the same. This is also sometimes called the *consistency* or *agreement* property of the blockchain. To eventually prove such safety guarantee of BCube, the following lemma first summarizes the properties of OverlayBB, whose proof is deferred to Appendix V:

Theorem 1. [guarantees of OverlayBB] *In Algorithm 1, if the committee has at least one honest member, then*

- *All honest nodes must return the same object.*
- *If the broadcaster is honest, then all honest nodes must return the object broadcast by the broadcaster.*

Finally, regardless of the committee, Algorithm 1 always returns within $2dm + s$ rounds.

Part of Theorem 1 requires the committee to contain some honest member. Consider any slot in epoch i. Recall that the committee for that slot is chosen using $\mathtt{beacon}_i$. This $\mathtt{beacon}_i$ is generated in epoch $i-1$, and may be biased and influenced by the adversary: i) the adversary may find multiple PoW solutions in epoch $i-1$, and cherry-pick the one that it likes; ii) if the τ broadcasters in the first τ slots of epoch $i-1$ are all malicious, then the adversary can predict the PoW challenge before time T_1, and can pre-compute many PoW solutions; iii) if the honest nodes fail to find any PoW solution in epoch $i-1$ or if the τ broadcasters in the τ candidate-holding slots in epoch $i-1$ are all malicious, then $\mathtt{beacon}_{i-1}$ may be reused as $\mathtt{beacon}_i$, and $\mathtt{beacon}_{i-1}$ may already be biased; iv) if the committee for some slot in epoch $i-1$ contains no honest members, then the honest nodes may not even agree on the PoW challenges and on what $\mathtt{beacon}_i$ is.

We will later reason about the probabilities of various random events, such as whether the committee in Theorem 1 contains some honest member. The adversary may influence such probabilities, by for example, biasing the beacons as explained above. The amount of such influence will depend on what strategy the adversary uses. We will carefully ensure that all our analyses (e.g., regarding the probabilities) hold, even under *the worst-case adversary that uses the optimal strategy*. In particular, our analyses will not make claims such as $\Pr[X] = y$, but only make claims such as $\Pr[X] \leq y$. This just means that while $\Pr[X]$ may be different under different strategies of the adversary, it can never be above y.

We now introduce some random variables. Consider all the slots in the blockchain. Let ρ be the number of slots in each epoch. For all integer $j \in [1, \infty)$ and all $\lambda \geq 1$, let random variable $\mathcal{Z}_\lambda(j)$ denote the event that all of the following events happen in the execution of BCube:

- For each slot j' where $1 \leq j' \leq j$, the committee for that slot contains at least one honest member.
- For each epoch i' where $1 \leq i' \leq \lceil \frac{j+1}{\rho} \rceil - 1$, no more than λ different PoW solutions are seen by honest nodes in epoch i'.

For all $\lambda \geq 1$, define $\mathcal{Z}_\lambda(0)$ to be an event that always occurs.

Roughly speaking, $\mathcal{Z}_\lambda(j)$ means that the execution is "good" up to slot j. The first part in $\mathcal{Z}_\lambda(j)$ corresponds to the requirement in Theorem 1, and the second part serves to facilitate later reasoning about $\Pr[\mathcal{Z}_\lambda(j)]$ via a recursion. In this second part, the λ solutions "seen by honest nodes" can be i) PoW solutions for epoch i' found by honest nodes, ii) PoW solutions for epoch i' found by malicious nodes in epoch i', and iii) PoW solutions for epoch i' found by malicious nodes before epoch i' started (if the PoW challenge is not fresh).

We now formally state the safety guarantee of BCube:

Theorem 2. [safety guaranteed in "good" execution] *For any given $\lambda \geq 1$ and $j \geq 1$, if $\mathcal{Z}_\lambda(j)$ occurs, then for each j' where $1 \leq j' \leq j$, all honest nodes in* BCube *must always have the same block in slot j' once the* OverlayBB *invocation for slot j' has completed.*

Proof. Consider any given j' where $1 \leq j' \leq j$. Then $\mathcal{Z}_\lambda(j)$ occurring means that the committee for slot j' has some honest member. For any given node, the j'-th block in its blockchain is simply the return value of the OverlayBB invocation on that node for the j'-th slot. By Theorem 1, such return value must be the same on all honest nodes. ☐

Theorem 2 guarantees the safety of BCube in "good" executions, but does not tell us the likelihood of the execution being "good". Theorem 3 next shows that conditioned upon the execution being "good" up to slot $j-1$, with high probability, it continues to be "good" up to slot j. Theorem 3 is based on the following parameterization of BCUBE: We set T_1 through T_5 to match the respective durations in Figure 3, and we set the weak PoW difficulty so that the honest nodes on expectation obtain two PoW solutions from T_2 to T_3. Changing these parameters will only affect the two constants "0.86" and "807" in the theorem. Also, Theorem 3 assumes that the adversary cannot adaptively corrupt honest nodes. Appendix I will explain that the negative effect of adaptive corruption is easily bounded, as long as the adaptivity is sufficiently "mild".

Theorem 3. ["good" execution occurs w.h.p.] *Consider any constant $f \leq 0.99$, and any positive integers λ and j. If $\Pr[\mathcal{Z}_\lambda(j-1)] > 0.9$, then conditioned upon $\mathcal{Z}_\lambda(j-1)$, we must have:*[7]

$$\begin{aligned}\Pr[\mathcal{Z}_\lambda(j)] &\geq 1 - \tfrac{\lambda f^m}{0.9(0.86-f^\tau)} - \tfrac{\lambda f^\tau}{0.9(0.86-f^\tau)} - \mathtt{Pois}(807,\lambda) \\ &= 1 - \lambda e^{-\Omega(m)} - \lambda e^{-\Omega(\tau)} - e^{-\Omega(\lambda)}\end{aligned}$$

Here $\mathtt{Pois}(807, \lambda)$ is defined to be $\Pr[X > \lambda]$, where X follows a Poisson distribution with mean 807.

Proof. See Appendix I. □

Asymptotically, the error probability in the above theorem is exponentially small with respect to m, τ, and λ, which can all be viewed as security parameters. As a concrete example, with $f = 0.7$, $\lambda = 1000$, $m \geq 79$, and $\tau \geq 91$, the above theorem gives[8] $\Pr[\mathcal{Z}_\lambda(j)] \geq 1 - 2^{-30}$. Hence we use a committee size of $m = 80$ in our later experiments when $f = 0.7$.

B. Liveness and Throughput of BCUBE

For any given slot, *liveness* of BCUBE means that BCUBE should always eventually confirm a block for that slot. *Throughput* simply equals block size times the average number of blocks confirmed per second. In some sense, throughput captures the "rate of liveness", in terms of the number of bits confirmed per second. Note that each slot in BCUBE has a corresponding OVERLAYBB invocation, which starts at a pre-determined time. The following theorem shows that once the invocation starts, within some well-defined time, we will have a confirmed block in that slot:

Theorem 4. [liveness guaranteed] *At most $2dm + s$ rounds (or $(2dm + s)\delta$ time with δ being the round duration) after the start of the corresponding* OVERLAYBB *invocation for a given slot in* BCUBE*, all honest nodes in* BCUBE *must have a confirmed block in that slot.*

Proof. Trivially follows from the fact that Algorithm 1 has exactly $2dm + s$ rounds. □

Due to space constraint, we defer our throughput analysis of BCUBE to Appendix II. Appendix II first derives an upper bound on the total number of bits that each honest node needs to send in each round. This upper bound will hold under *all possible strategies of the adversary and all possible randomness outcomes*. Using this upper bound, Appendix II then shows that, under practical parameters, BCUBE has a throughput of $\mathbb{T} \approx \frac{\mathbb{B}}{2w} = \Theta(\frac{\mathbb{B}}{w})$ and a TTB ratio of $\mathbb{R} \approx \Theta(\frac{1}{w})$.

VIII. Implementation and Experimental Results

Implementation. We have implemented BCUBE in Go and using TCP, except the following parts that have no effects on our experimental results: Since beacon generation from the weak PoW takes one epoch (e.g., one day), we did not implement the weak PoW or propagate the PoW solutions. (Propagating the PoW solutions has negligible cost, since each node only sends/relays one 20-byte PoW solution in each epoch.) We instead directly inject a random beacon. We still properly determine various parameters, such as committee size, based on our weak PoW design. We did not implement transactions, and we fill each block with random bits. There is no stake transfer, and each node always holds one stake (coin). Finally, we will run up to 500 BCUBE nodes on each physical machine. Due to CPU constraint, we did not implement aggregate signature signing/validation, and also did not implement secure hash function. We replace all of these with dummy functions. Appendix III will show, via a careful calculation, that *regardless of the strategy of the adversary and regardless of what messages the malicious nodes may send to the honest nodes*, under all settings in this section, in every second each honest BCUBE node only needs to do at most 152 aggregate signature signing/validation operations, and at most 610 secure hashes (for Merkle proof verification). Similarly, due to memory constraint, the 500 BCUBE nodes on the same machine are implemented as separate threads in one Go process, instead of as 500 separate Go processes. Of course, these threads do not interact with each other via the shared heap space.

Experimental settings. We run our experiments on 21 high-end PCs, each with 10Gbps bandwidth, in a local-area network. The first PC runs a single BCUBE node (with the maximum degree of 42 — see later). Each of the remaining 20 PCs run 500 BCUBE nodes (with the last PC running 499 nodes), so that each BCUBE node has about 20Mbps bandwidth. Altogether, this gives us total 10000 BCUBE nodes. Running one BCUBE node on the first PC allows us to directly measure the total network traffic on the Ethernet interface of that PC in every second, using the Linux bandwidth monitoring tool `bmon`. Our measurement results in Appendix IV confirm that a BCUBE node (even with the maximum degree of 42) indeed never uses more than 20Mbps bandwidth.

We construct the overlay topology in a similar way as in [11], [33]: Each node A keeps choosing random nodes to establish (undirected) edges to, until it manages to establish 20 edges. To prevent A from forming edges to the other nodes on the same machine as A and hence bypassing the network, the random nodes are chosen from all the nodes on the other machines. To avoid having too many neighbors, each node stops accepting new edges after it has accepted 22 edges from

[7]We use Poisson distribution to approximate binomial distributions here.

[8]Conceptually, Theorem 2 focuses on the error probability of a given committee (i.e., for the jth slot) in BCUBE. This is consistent with other analysis in the literature [18], [19], [21], [35]. If needed, one can easily translate such guarantees to the entire execution.

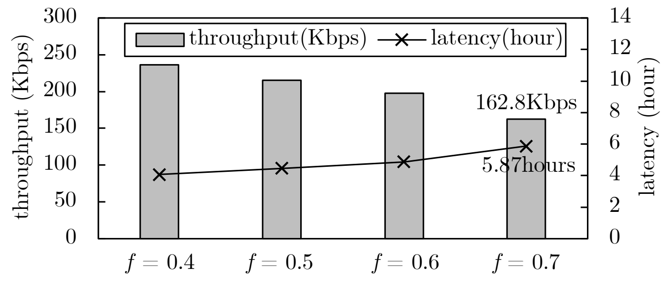


Fig. 4: End-to-end performance of BCUBE.

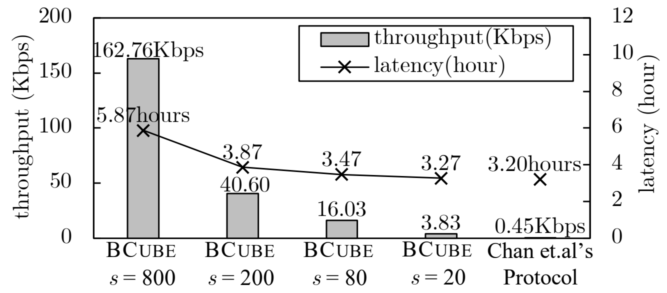


Fig. 5: Comparison of our protocol BCUBE and the state-of-the-art Chan et al.'s protocol [6].

other nodes. Hence the node degrees range from 20 to 42, with the average being 40. We set $\delta_1 = 10$s, $\delta_2 = 2$s, and $\delta = 12$s. We assume that all edges are good in our experiments. While we do not explicitly emulate wide-area message propagation delay, we expect such delay to be typically well below our δ_1 value of 10 seconds. (The messages in our experiments always have size no larger than 10KB.) We observe that with the above construction, the honest subgraph typically has diameter of no more than 6 (even if we uniformly randomly choose 0.7 fraction of the nodes to be malicious). Hence we assume $d = 6$ in our experiments.

We consider f ranging from 0.4 to 0.7. Since BCUBE focuses on malicious majority, we do not consider smaller f values. Recall that larger s (i.e., number of fragments) gives higher throughput but longer confirmation latency. To strike a balance, our experiments always use $s = 800$. To achieve an error probability $\epsilon \leq 2^{-30}$, and following Theorem 3, we use $m = 35$, 45, 55, and 80, for $f = 0.4$, 0.5, 0.6, and 0.7, respectively. We use a block size of 2MB in BCUBE, and an inter-block time of roughly 68, 74, 81, and 98 seconds for $f = 0.4$, 0.5, 0.6, and 0.7, respectively. These parameters are chosen such that based on the analysis in Appendix II, each node consumes no more than about 90% of its 20Mbps available bandwidth (even in the very worst-case). Note that here we use the exact version of the analysis in Appendix II, without applying any approximation such as $\mathbb{Y} \approx \frac{wl}{s}$.

End-to-end performance. Figure 4 plots BCUBE's transaction throughput and confirmation latency. As expected, the confirmation latency increases with f, since larger f entails a larger committee size (m) and in turn more rounds in OVERLAYBB. Similarly, the transaction throughput decreases with larger f since as each invocation of OVERLAYBB takes longer to finish, we need to correspondingly increase the inter-block time. This then decreases throughput. Nevertheless, even when $f = 0.7$, BCUBE still achieves a throughput of about 163Kbps and a confirmation latency of less than 6 hours. As explained in Section I where we used Bitcoin as a reference point, such performance is already "practically usable": Bitcoin entails a confirmation latency of about 9.3 hours to achieve $\epsilon \leq 2^{-30}$ under $f = 0.25$, and Bitcoin's throughput is about 14Kbps.

Compare with state-of-the-art design. There has been rather limited amount of prior work on designing blockchains for tolerating $f \geq 0.5$. The current state-of-the-art approach is via Chan et al.'s protocol [6]. Strictly speaking, Chan et al.'s protocol is a byzantine broadcast protocol, instead of a blockchain. But one could use Chan et al.'s protocol as the core to build a blockchain, in the same way as we use OVERLAYBB to build BCUBE. To enable a direct comparison, we take BCUBE, and then replace OVERLAYBB with Chan et al.'s protocol (as described in Section III), to obtain a blockchain based on their protocol. We use our own implementation of their protocol, since their work does not have implementation. Section IV-A explained that when running on multi-hop topologies, Chan et al.'s protocol would require infinite bandwidth if a malicious broadcaster keeps injecting conflicting messages. Our experiments for their protocol explicitly assume away this particular adversarial behavior — doing so only makes the results for their protocol better.

Due to space constraints, we only present our results on Chan et al.'s protocol for $f = 0.7$. Specifically, we measure the throughput/latency of Chan et al.'s protocol (i.e., the resulting blockchain), under the same setting as our BCUBE, such as 12-second round duration, same topology, around 98-second inter-block time, and a committee size of 80. We also use pipelined invocations for Chan et al.'s protocol, in the same way as we do in our protocol. We set the block size to be such that in all cases (including all adversarial strategies and randomness), the bandwidth consumed by each node is no more than 90% of the 20Mbps available bandwidth, which is the same constraint we imposed in the experiments of our BCUBE. Under such a constraint, the maximum block size we can use in the experiments for Chan et al.'s protocol is about 5.5KB.

Figure 5 compares the performance of our protocol and Chan et al.'s protocol, as observed in our experiments. Chan et al.'s protocol achieves a throughput of about 0.45Kbps, while BCUBE achieves about 163Kbps, which is over 350 times higher. Such large improvement primarily comes from the fact that BCUBE/OVERLAYBB breaks each block into $s - 1 = 799$ fragments and can delay the forwarding of individual fragments whenever needed, to avoid congestion in any given round. With some over-simplification, Chan et al.'s protocol can be viewed as having only a single fragment. This is also why their protocol can only use 5.5KB block size, while we can support 2MB block size.

Using many fragments in BCUBE does increase the latency:

Chan et al.'s protocol has a latency of 3.20 hours, while ours is 5.87 hours. To gain more insights, Figure 5 further presents the performance of BCUBE when using fewer fragments, with $s = 200$, $s = 80$, and $s = 20$. In particular, with $s = 20$ fragments, our latency is 3.27 hours, which is only 2.2% larger than their latency. Yet with $s = 20$, we still achieve more than 850% of the throughput of their protocol, and can support block size of about 47KB. Hence even if BCUBE is forced to provide almost the same latency as Chan et al.'s protocol, BCUBE still provides significantly higher throughput.

IX. RELATED WORKS

Byzantine broadcast. Being a classic distributed computing problem, byzantine broadcast has been extensively studied. We will only focus on byzantine broadcast protocols [6], [8], [9], [14], [25], [31], [32] that can tolerate $f \geq \frac{1}{2}$. Most of these are actually theoretical designs without implementation. Section I and III already discussed [6], [8], [31]. The protocols from [9], [14], [25] all require direct point-to-point communication on a clique, and hence does not work for multi-hop topologies. Furthermore, these protocols are designed for a permissioned setting with a fixed set of n nodes. The following nevertheless still reviews the techniques used in [9], [14], [25], and characterizes their TTB ratios.

Hirt and Raykov's protocol [14] breaks the object into n fragments, each with $\frac{l}{n}$ size, to optimize for communication complexity. For each fragment and each node, they invoke a smaller black-box byzantine broadcast protocol, resulting in total n^2 sequential invocations. Doing so enables later invocations to benefit from information collected during earlier invocations. The protocol takes total at least n^2 rounds. In some rounds, a node needs to send one fragment (i.e., $\frac{l}{n}$ bits). Hence the maximum l the protocol can support, given $\mathbb{B}$ available bandwidth, is $l_0 = \mathbb{B}\delta n$. We thus have $\mathbb{T} \leq \frac{l_0}{n^2\delta} = \frac{\mathbb{B}\delta n}{n^2\delta} = \frac{\mathbb{B}}{n}$ and $\mathbb{R} = \mathbb{T}/\mathbb{B} \leq \frac{1}{n}$. Ganesh and Patra's protocol [9] improves upon [14], and reduces the time complexity to about n rounds. In [9], some rounds are used for propagating the fragments. In each such round, a node may need to send its fragment to up to n nodes, incurring $n \times \frac{l}{n} = l$ bits of communication. Hence the maximum l the protocol can support, given $\mathbb{B}$ available bandwidth, is $l_0 = \mathbb{B}\delta$. We thus have $\mathbb{T} \leq \frac{l_0}{n\delta} = \frac{\mathbb{B}}{n}$ and $\mathbb{R} = \mathbb{T}/\mathbb{B} \leq \frac{1}{n}$. In comparison to [9], [14], OVERLAYBB also breaks an object into fragments, but for a different purpose of improving throughput. Because of this, most issues in OVERLAYBB such as delaying and compensation are not relevant to [9], [14].

Nayak et al.'s protocol [25] further improves the communication complexity of [9]. In their protocol, instead of sending the object to all other nodes directly, a node uses erasure coding and sends one fragment to each of the n nodes. The n nodes will then each forward its received fragment to all other nodes. Each fragment has size at least $\frac{l}{n}$, and hence a node needs to send at least $n \cdot \frac{l}{n} = l$ bits in some rounds. The maximum l the protocol can support is then $l_0 = \mathbb{B}\delta$. Their protocol has total $fn + 1$ rounds. This leads to $\mathbb{T} \leq \frac{l_0}{(fn+1)\delta}$ and $\mathbb{R} = \mathbb{T}/\mathbb{B} < \frac{1}{fn}$. Their idea of using erasure coding is largely orthogonal to the techniques in OVERLAYBB.

Finally, Wan et al. [32] recently propose a constant-round byzantine broadcast protocol for tolerating $f \geq \frac{1}{2}$. When adapted to our multi-hop setting, their protocol takes at least d rounds and in each round, a node may need to send the l-bit object to its w neighbors. Hence the maximum l the protocol can support is $l_0 = \frac{\mathbb{B}\delta}{w}$. In turn, $\mathbb{R} = \mathbb{T}/\mathbb{B} = \frac{l_0}{d\delta}/\mathbb{B} = \Theta(\frac{1}{dw})$. In comparison, our OVERLAYBB has $\mathbb{R} = \Theta(\frac{1}{w})$. More importantly, their protocol further needs each node to send up to n^2 bits or more (for additional protocol information) to each of its w neighbors. Hence their protocol achieves $\mathbb{R} = \Theta(\frac{1}{dw})$ only when l reaches the order of wn^2, which translates to about 500MB under our experimental parameters. The block size in blockchains is typically much smaller than 500MB.

Blockchains. Most existing blockchains (e.g., [1], [3], [7], [11], [16], [19], [21], [33], [35]) today can only tolerate $f < \frac{1}{2}$. By leveraging the "reputation" of the nodes, RepuCoin [34] can tolerate temporary malicious majority — namely, temporary spikes in f (but not $f \geq \frac{1}{2}$ in general). While blockchains can be built from byzantine broadcast, and hence tolerate $f \geq \frac{1}{2}$, this fact has been largely neglected in the literature. Pass and Shi [26] mention the design of a blockchain based on the Dolev-Strong protocol [8] for byzantine broadcast. As explained in Section I, using the Dolev-Strong protocol will result in rather low throughput (e.g., 0.072Kbps). Our contribution is exactly to overcome this central issue. Note that the main focus of [26] is not on tolerating $f \geq \frac{1}{2}$, but on providing fast transaction confirmation when a super majority of the users are honest. In addition, their work is mainly theoretical, with no implementation.

X. CONCLUSIONS

We have presented BCUBE, the very first blockchain that can tolerate $f \geq \frac{1}{2}$, while achieving practically usable transaction throughput and latency. At the core of BCUBE is our novel byzantine broadcast protocol OVERLAYBB, which can achieve significantly better throughput than prior protocols. BCUBE still leaves many questions unanswered. For example, can we further improve its performance? Can we generalize beyond Proof-of-Stake? Can we offer progressive confirmation, as in Bitcoin, so that a transaction's likelihood of being confirmed grows with time, even before it is fully-confirmed? All these are interesting open questions for future research.

ACKNOWLEDGMENT

We thank the anonymous IEEE Security & Privacy reviewers for their detailed and helpful comments on this paper.

DISCLOSURE BY AUTHORS

Haifeng Yu is an Associate Professor in School of Computing, National University of Singapore (NUS). Haifeng is also a Co-PI of NUS CRYSTAL Centre, which is a blockchain-related research centre. Prateek Saxena is an Associate Professor in School of Computing, NUS. Prateek is also a Co-Director of NUS CRYSTAL Centre, and a co-founder of Zilliqa Research, which is related to blockchains.

References

[1] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and A. Spiegelman, "Solida: A blockchain protocol based on reconfigurable byzantine consensus," in *International Conference on Principles of Distributed Systems*, 2017.

[2] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in *IEEE Symposium on Security and Privacy*, 2017.

[3] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, "Prism: Deconstructing the Blockchain to Approach Physical Limits," in *CCS*, 2019.

[4] S. Bojja Venkatakrishnan, G. Fanti, and P. Viswanath, "Dandelion: Redesigning the bitcoin network for anonymity," in *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2017.

[5] D. Boneh, M. Drijvers, and G. Neven, "Compact multi-signatures for smaller blockchains," in *ASIACRYPT*, 2018.

[6] T.-H. H. Chan, R. Pass, and E. Shi, "Sublinear-round byzantine agreement under corrupt majority," in *IACR International Conference on Public-Key Cryptography*, 2020.

[7] P. Daian, R. Pass, and E. Shi, "Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake," in *International Conference on Financial Cryptography and Data Security*, 2019.

[8] D. Dolev and H. R. Strong, "Authenticated algorithms for byzantine agreement," *SIAM Journal on Computing*, vol. 12, no. 4, pp. 656–666, 1983.

[9] C. Ganesh and A. Patra, "Broadcast extensions with optimal communication and round complexity," in *PODC*, 2016.

[10] J. Garay and A. Kiayias, "Sok: A consensus taxonomy in the blockchain era," in *Cryptographers' Track at the RSA Conference*, 2020.

[11] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *SOSP*, 2017.

[12] S. Haig, "Bitcoin cash could face 51% attack for $10,000 in rented hashpower," https://cointelegraph.com/news/bitcoin-cash-could-face-51-attack-for-10-000-in-rented-hashpower, 2020.

[13] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *USENIX Security Symposium*, 2015.

[14] M. Hirt and P. Raykov, "Multi-valued byzantine broadcast: The $t < n$ case," in *ASIACRYPT*, 2014.

[15] R. Hou, H. Yu, and P. Saxena, "Using throughput-centric byzantine broadcast to tolerate malicious majority in blockchains," 2021, arxiv preprint, arXiv:2108.01341.

[16] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *CRYPTO*, 2017.

[17] L. Kiffer, R. Rajaraman, and abhi shelat, "A Better Method to Analyze Blockchain Consistency," in *CCS*, 2018.

[18] E. 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 *USENIX Security Symposium*, 2016.

[19] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, and B. Ford, "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding," in *IEEE Symposium on Security and Privacy*, 2018.

[20] Y. Kwon, J. Liu, M. Kim, D. Song, and Y. Kim, "Impossibility of full decentralization in permissionless blockchains," in *ACM Conference on Advances in Financial Technologies*, 2019.

[21] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *CCS*, 2016.

[22] N. A. Lynch, *Distributed algorithms*. Elsevier, 1996.

[23] Y. Marcus, E. Heilman, and S. Goldberg, "Low-resource eclipse attacks on ethereum's peer-to-peer network." *IACR Cryptol. ePrint Arch.*, 2018.

[24] J. Martin, "Bitcoin gold blockchain hit by 51% attack leading to $70k double spend," https://cointelegraph.com/news/bitcoin-gold-blockchain-hit-by-51-attack-leading-to-70k-double-spend, 2020.

[25] K. Nayak, L. Ren, E. Shi, N. H. Vaidya, and Z. Xiang, "Improved extension protocols for byzantine broadcast and agreement," *arXiv preprint arXiv:2002.11321*, 2020.

[26] R. Pass and E. Shi, "Thunderella: Blockchains with optimistic instant confirmation," in *EUROCRYPT*, 2018.

[27] J. Redman, "Small ethereum clones getting attacked by mysterious '51 crew'," https://news.bitcoin.com/ethereum-clones-susceptible-51-attacks/, 2016.

[28] Rocky, "Krypton recovers from a new type of 51% network attack," https://cryptohustle.com/krypton-recovers-from-a-new-type-of-51-network-attack/, 2016.

[29] D. Topkis, "Concurrent broadcast for information dissemination," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 10, 1985.

[30] M. Tran, I. Choi, G. J. Moon, V.-A. Vu, and M. S. Kang., "A stealthier partitioning attack against bitcoin peer-to-peer network," in *IEEE Symposium on Security and Privacy*, 2020.

[31] G. Tsimos, J. Loss, and C. Papamanthou, "Nearly quadratic broadcast without trusted setup under dishonest majority," *IACR Cryptology ePrint Archive*, 2020.

[32] J. Wan, H. Xiao, E. Shi, and S. Devadas, "Expected constant round byzantine broadcast under dishonest majority." *IACR Cryptology ePrint Archive*, 2020.

[33] H. Yu, I. Nikolic, R. Hou, and P. Saxena, "OHIE: Blockchain Scaling Made Simple," in *IEEE Symposium on Security and Privacy*, 2020.

[34] J. Yu, D. Kozhaya, J. Decouchant, and P. Esteves-Verissimo, "Repucoin: Your reputation is your power," *IEEE Transactions on Computers*, vol. 68, no. 8, pp. 1225–1237, 2019.

[35] M. Zamani, M. Movahedi, and M. Raykova, "RapidChain: Scaling Blockchain via Full Sharding," in *CCS*, 2018.

Appendix I: Proof for Theorem 3

Theorem 3 (Restated)**. ["good" execution occurs w.h.p.]** *Consider any constant $f \leq 0.99$, and any positive integers λ and j. If $\Pr[\mathcal{Z}_\lambda(j-1)] > 0.9$, then conditioned upon $\mathcal{Z}_\lambda(j-1)$, we must have:*[9]

$$\begin{aligned}\Pr[\mathcal{Z}_\lambda(j)] &\geq 1-\tfrac{\lambda f^m}{0.9(0.86-f^\tau)}-\tfrac{\lambda f^\tau}{0.9(0.86-f^\tau)}-\texttt{Pois}(807,\lambda)\\ &= 1-\lambda e^{-\Omega(m)}-\lambda e^{-\Omega(\tau)}-e^{-\Omega(\lambda)}\end{aligned}$$

Here $\texttt{Pois}(807,\lambda)$ is defined to be $\Pr[X > \lambda]$, where X follows a Poisson distribution with mean 807.

Proof. All probabilities in this proof, unless otherwise mentioned, are conditioned upon $\mathcal{Z}_\lambda(j-1)$. We only prove the harder case where $j \geq 2$ and $\lceil \frac{j}{\rho} \rceil \neq \lceil \frac{j+1}{\rho} \rceil$. (In other cases, the second part in $\mathcal{Z}_\lambda(j)$ trivially follows from the second part in $\mathcal{Z}_\lambda(j-1)$, and hence the proof is similar but easier.) Let $i = \lceil \frac{j+1}{\rho} \rceil - 1$. We define several random events:

- $\mathcal{W}_1$: The committee for slot j contains at least one honest member.
- $\mathcal{W}_2$: Among the first τ slots of epoch i, where each slot has a corresponding committee and broadcaster, there exists at least one slot whose broadcaster is honest.
- $\mathcal{W}_3$: From time T_1 through T_5 in epoch i, the honest nodes and the adversary combined find no more than λ PoW solutions. (This does not include PoW solutions found by the adversary prior to T_1, for example, when the PoW challenge is not fresh.)

We will later prove that:

$$\Pr[\mathcal{W}_1] \quad \geq \quad 1-\frac{\lambda f^m}{0.9\times(0.86-f^\tau)} \qquad (1)$$

$$\Pr[\mathcal{W}_2] \quad \geq \quad 1-\frac{\lambda f^\tau}{0.9\times(0.86-f^\tau)} \qquad (2)$$

$$\Pr[\mathcal{W}_3] \quad \geq \quad 1-\texttt{Pois}(807,\lambda) \qquad (3)$$

Hence with probability at least $1-\frac{\lambda f^m}{0.9(0.86-f^\tau)}-\frac{\lambda f^\tau}{0.9(0.86-f^\tau)}-\texttt{Pois}(807,\lambda)$, all three events occur. Recall that the PoW challenge in epoch i is the concatenation of all the nonces in

[9] We use Poisson distribution to approximate binomial distributions here.

the first τ slots. By events $\mathcal{Z}_\lambda(j-1)$, $\mathcal{W}_1$, $\mathcal{W}_2$, and Theorem 1, we have that i) all honest nodes agree on the PoW challenge in epoch i, and ii) the PoW challenge in epoch i is fresh in the sense that the adversary does not see the challenge before T_1. Together with event $\mathcal{W}_3$, this means that no more than λ PoW solutions are seen by honest nodes in epoch i, which we define as event $\mathcal{W}_4$. Finally, $\mathcal{Z}_\lambda(j)$ follows directly from $\mathcal{W}_1$, $\mathcal{W}_4$, and $\mathcal{Z}_\lambda(j-1)$.

In the following, we analyze $\Pr[\mathcal{W}_1]$, $\Pr[\mathcal{W}_2]$, and $\Pr[\mathcal{W}_3]$. We start with $\Pr[\mathcal{W}_1]$. Consider any given i' where $1 \le i' \le i-1$, and PoW solution x seen by some honest node in epoch i'. Let $y = \mathtt{hash}_2(\mathtt{challenge}|x)$, where $\mathtt{challenge}$ is the PoW challenge corresponding to x. Essentially, y is a potential beacon value for epoch $i'+1$, and may potentially further be reused later in epoch i. If we choose the committee by using $\mathtt{hash}_1(\text{slot number}|y)$ as randomness, then the probability of the committee containing no honest member is at most f^m.

Next, we upper bound the probability that y is used/reused as the beacon in epoch i. Define $Z_1 = \mathcal{Z}_\lambda(i'\rho)$ and $Z_2 = \mathcal{Z}_\lambda(j-1)$. Conditioned upon Z_1 only, define $A_{i'}$ to be the random event where for every $g \in [i'+1, i-1]$, epoch g satisfies at least one of the following two conditions: i) no honest node finds any PoW solution in epoch g, or ii) if y were used as the beacon in epoch g, then none of the τ broadcasters in the τ candidate-holding blocks in epoch g would be honest. Note that $A_{i'}$ is well-defined, even if Z_2 does not happen, and even if the honest nodes do not agree on the PoW challenge in epoch g: In those cases, the first condition in $A_{i'}$ simply means that no honest node solves the PoW, based on whatever each honest node individually believes to be the PoW challenge. Define $p(i')$ to be the probability of $A_{i'}$ happening, conditioned upon Z_1 only. Define $q(i')$ to be the probability of $A_{i'}$ happening, conditioned upon Z_2. With a Poisson approximation and since the honest nodes on expectation find two PoW solutions in each epoch, we have $p(i') \le (f^\tau + 0.14)^{i-i'-1}$. In turn, by Bayes' formula and since Z_2 implies Z_1, we have $q(i') = \frac{\Pr[A_{i'}Z_2]}{\Pr[Z_2]} = \frac{\Pr[A_{i'}Z_1Z_2]}{\Pr[Z_2]} \le \frac{\Pr[A_{i'}Z_1]}{\Pr[Z_2]} = \frac{\Pr[A_{i'}|Z_1]\Pr[Z_1]}{\Pr[Z_2]} \le \frac{\Pr[A_{i'}|Z_1]}{\Pr[Z_2]} = \frac{p(i')}{\Pr[\mathcal{Z}_\lambda(j-1)]} \le \frac{(f^\tau+0.14)^{i-i'-1}}{0.9}$. Now conditioned upon $\mathcal{Z}_\lambda(j-1)$, in order for y to be used as the beacon in epoch i, the event $A_{i'}$ must happen. Hence the probability of y being used as the beacon in epoch i is at most $\frac{(f^\tau+0.14)^{i-i'-1}}{0.9}$.

The probability of y being used as the beacon in epoch i and further causing the committee for slot j to not contain any honest member is then at most $f^m \times \frac{(f^\tau+0.14)^{i-i'-1}}{0.9}$. Finally, there are at most λ different y values in each epoch $i' \in [1, i-1]$, and we need to take a union bound over all those. Hence we have $\Pr[\mathcal{W}_1] \ge 1 - \sum_{i'=1}^{i-1}(\lambda f^m \frac{(f^\tau+0.14)^{i-i'-1}}{0.9}) \ge 1 - \frac{\lambda f^m}{0.9(0.86-f^\tau)}$.

We move on to $\Pr[\mathcal{W}_2]$. Each of the first τ slots in epoch i has a corresponding broadcaster. $\mathcal{W}_2$ essentially is the event that at least one of these τ broadcasters is honest. Following similar reasoning as above, we have $\Pr[\mathcal{W}_2] \ge 1 - \sum_{i'=1}^{i-1}(\lambda f^\tau \frac{(f^\tau+0.14)^{i-i'-1}}{0.9}) \ge 1 - \frac{\lambda f^\tau}{0.9(0.86-f^\tau)}$.

Finally we consider $\Pr[\mathcal{W}_3]$. Since the adversary has at most 100 times the computational power as honest nodes, and since the honest nodes on expectation find 2 PoW solutions from T_2 to T_3, one can verify that on expectation the adversary finds no more than 805 solutions from T_1 to T_5. Hence $\Pr[\mathcal{W}_3] \ge 1 - \mathtt{Pois}(2+805, \lambda) = 1 - \mathtt{Pois}(807, \lambda)$. $\square$

Remark. Theorem 3 assumes that the adversary cannot adaptively corrupt honest nodes. If adaptive corruption is possible, then the adversary may corrupt the committee members after seeing the beacon and before the committee has done its work. Assume that the adversary takes at least x epochs to adaptively corrupt nodes. Then for a given slot in epoch i and following a similar reasoning as in the proof of Theorem 3, the probability of the adversary adaptively corrupting the committee members in that slot is at most $\sum_{i'=1}^{i-x+1}(\lambda \frac{(f^\tau+0.14)^{i-i'-1}}{0.9}) \le \frac{\lambda(f^\tau+0.14)^{x-2}}{0.9(0.86-f^\tau)} \approx \lambda \cdot 0.14^{x-2}$, which drops exponentially with x. If needed, the constant 0.14 can be further decreased as well, by setting the PoW easier (and increasing m, τ, and λ accordingly).

Appendix II: Throughput Analysis of BCube

This section analyzes the throughput of BCube. The throughput of BCube follows from the throughput of all the OverlayBB invocations. For any given OverlayBB invocation, define $\mathbb{Y}$ to be the maximum number of bits that an honest node needs to send in a round, with the maximum taken across all honest nodes, all rounds, all possible strategies of the adversary, and all possible randomness. Intuitively, $\mathbb{Y}$ is the very worst-case number of bits a node needs to send in a round. We derive $\mathbb{Y}$ first, and then derive throughput.

From the pseudo-code of Algorithm 1 through 3, one can easily see that in each round, an honest node only sends messages at Line 44, 54, and 67, *regardless of the attack strategy of the adversary and regardless of the randomness*. Furthermore, all these messages are always of fixed size. Let $l_{\mathtt{nonce}}$, $l_{\mathtt{hash}}$, and $l_{\mathtt{sig}}$ be the size of a nonce, a hash, and an aggregate signature in Algorithm 1, respectively. Also recall that each message is sent to all the neighbors of the node, and each node has at most w neighbors. Straight-forward counting shows that the total number of bits sent by each honest node in each round in one OverlayBB invocation is at most $\mathbb{Y} = w \times (2 \times (l_{\mathtt{hash}} + l_{\mathtt{sig}} + m) + \max(\lceil \frac{l}{s-1} \rceil + (l_{\mathtt{hash}} + 1) \cdot \lceil \log_2 s \rceil,\ l_{\mathtt{nonce}} + (l_{\mathtt{hash}} + 1) \cdot \lceil \log_2 s \rceil + l_{\mathtt{sig}} + m))$, since:

- At Line 44, the total size of r and $\sigma(r)$ is at most $l_{\mathtt{hash}} + l_{\mathtt{sig}} + m$.
- At Line 54, the size of x_i (including the Merkle proof and the index i) is $\lceil \frac{l}{s-1} \rceil + (l_{\mathtt{hash}} + 1) \cdot \lceil \log_2 s \rceil$.
- At Line 67, the total size of x_s and $\sigma(x_s)$ is at most $l_{\mathtt{nonce}} + (l_{\mathtt{hash}} + 1) \cdot \lceil \log_2 s \rceil + l_{\mathtt{sig}} + m$.

Under practical settings, including our experimental settings later, the term $\lceil \frac{l}{s-1} \rceil$ is usually significantly (e.g., 10 times) larger than the other terms, and the value s is usually not too small (e.g., ≥ 20). In such cases, we simply have $\mathbb{Y} \approx \frac{wl}{s}$.

We now derive throughput. Let $\mathbb{B}$ be the available bandwidth on each node. Recall that δ is the round duration, and let γ

denote the number of pipelined invocations of OVERLAYBB that each node has at any point of time. Let $l = l_0$ be the solution for the equation $\mathbb{Y} \times \gamma = \mathbb{B} \times \delta$ (namely, the bandwidth needed in each round equals the bandwidth available). Each OVERLAYBB invocation can thus confirm a block of size l_0 every $(2dm + s)\delta$ time. The total throughput is then $\mathbb{T} = \frac{\gamma l_0}{(2dm+s)\delta}$. Under the approximation of $\mathbb{Y} \approx \frac{wl}{s}$, we have $l_0 \approx \frac{s\mathbb{B}\delta}{w\gamma}$ and $\mathbb{T}/\mathbb{B} \approx \frac{\gamma s \mathbb{B}\delta}{(2dm+s)\delta w \gamma}/\mathbb{B} = \frac{s}{(2dm+s)w}$. Setting $s = 2dm$ gives $\mathbb{R} = \mathbb{T}/\mathbb{B} \approx \frac{1}{2w} = \Theta(\frac{1}{w})$ and $\mathbb{T} \approx \frac{\mathbb{B}}{2w}$.

Note that the above final throughput and TTB ratio is *independent* of γ, since eventually the γ term gets cancelled out. Hence our final results hold *regardless of* whether pipelined invocations are used (i.e., whether $\gamma \geq 2$ or $\gamma = 1$). This is expected: Using multiple pipelined invocations implies that each invocation gets only a portion of the available bandwidth and hence can only broadcast smaller-sized objects. These two factors, multiple invocations and smaller objects, cancel out. This is also consistent with the intuition that one cannot boost throughput, simply by using pipelined invocations.

To summarize, our analysis in this section shows that regardless of the attack strategy of the adversary, the total throughput of all the OVERLAYBB invocations in BCUBE is $\mathbb{T} \approx \frac{\mathbb{B}}{2w} = \Theta(\frac{\mathbb{B}}{w})$, under the approximation of $\mathbb{Y} \approx \frac{wl}{s}$.

APPENDIX III: NUMBER OF AGGREGATE SIGNATURE SIGNING/VALIDATION OPERATIONS AND SECURE HASH COMPUTATIONS

Via a careful calculation, this section shows that under all settings in Section VIII, *regardless of the strategy of the adversary and regardless of what messages the malicious nodes may send to the honest nodes*, in every second an honest BCUBE node only needs to do:

- Adding a signer to a aggregate signature: at most 55 times
- Aggregate signature verification (where the signature *passes* verification): at most 55 times
- Aggregate signature verification (where the signature *fails* verification): at most 42 times
- Merkle proof verification (where the proof *passes* verification): at most 19 times
- Merkle proof verification (where the proof *fails* verification): at most 42 times

Under all settings in Section VIII, each Merkle proof verification takes no more than 10 secure hashes. Hence in every second, each honest BCUBE node only needs to do at most $55 + 55 + 42 = 152$ aggregate signature signing/validation operations, and at most $(19 + 42) \times 10 = 610$ secure hash computations for Merkle proof verification. Note that all these numbers are *worst-case* numbers: The actual numbers can be even smaller, for example, when there is no active attack on BCUBE.

The following calculates the worst-case number of various operations. In any given round of OVERLAYBB, by the pseudo-code in Section V, a node adds a signer to an aggregate signature at most 3 times. Under all settings in Section VIII, we have no more than 217 pipelined invocations of OVERLAYBB at any given point of time. Since each round has 12 seconds, this translates to at most $3 \times 217/12 < 55$ times/second.

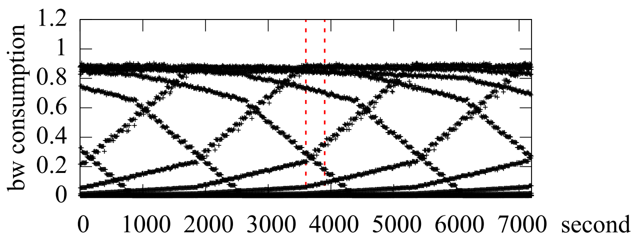
For verifying aggregate signatures, we use *lazy verification*: A node *only* verifies the signature on an item when it is about to use that item, instead of immediately upon receiving that item from its neighbors. For example in each round, a node may receive many Merkle roots from all its neighbors, but only picks the top two Merkle roots with the largest number of weighted signers, and processes those. The node will then simply verify the signatures (including the number of signers) on those two Merkle roots. If the signature does not pass verification, the node will pick again, until it gets two Merkle roots with valid signatures. One can then confirm, based on the pseudo-code in Section V, that in each round, a node does at most 3 aggregate signature verifications where the signature *passes* verification. Hence the rate of such verification is again at most 55 times/second. By similar reasoning, one can confirm, based on the pseudo-code, that a node does at most 19 Merkle proof verifications (where the proof *passes* verification) per second.

Finally, whenever an aggregate signature or Merkle proof does not pass verification, the neighbor who sent the corresponding item must be malicious. Hence a node *blacklists such a neighbor, and discards all previous/future messages from that neighbor*. With this simple trick, since each node has at most 42 neighbors in all our experiments, a node does at most 42 aggregate signature verifications where the signature *fails* verification, and at most 42 Merkle proof verifications where the proof *fails* verification.

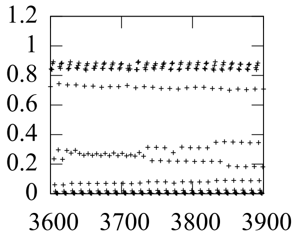
APPENDIX IV: SANITY CHECK ON THE BANDWIDTH CONSUMPTION OF BCUBE NODE

This section provides a sanity check on the bandwidth consumption of a BCUBE node in our experiments. Our goal is to verify that each BCUBE node indeed never uses more than 20Mbps bandwidth. Note that this does not directly follow from the 10Gbps aggregate available bandwidth across the 500 BCUBE nodes on one physical machine, since the 10Gbps may not be shared evenly. To do this sanity check, we pick an arbitrary node with the maximum degree of 42 (larger degree leads to more bandwidth consumption), and allocate a PC to run only that node. We then directly measure the total network traffic on the Ethernet interface of that PCs in every second, using the linux bandwidth monitoring tool `bmon`.

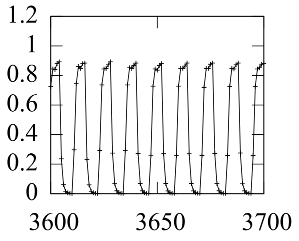
Figure 6(a)-(c) plot such measured bandwidth consumption under $f = 0.7$, as a fraction of 20Mbps. Results under other f values are similar. As expected, this fraction never exceeds 1.0, confirming that the node indeed never uses more than 20Mbps bandwidth. The zig-zag pattern in Figure 6(c) is also expected: Recall that each OVERLAYBB invocation has $2dm+s = 1760$ rounds. When there is no active attack, a node only needs to send messages in the first 800 rounds. Also recall that in these experiments, at any point of time, a node has many pipelined OVERLAYBB invocations. Based on such parameters, Figure 7 plots the computed number of invocations that need to send



(a) bandwidth consumption in every second



(b) zoomed in from above



(c) zoomed in from above

Fig. 6: Bandwidth consumption of an BCUBE node as a fraction of 20Mbps. As expected, this fraction never exceeds 1.0. Figure 6(a) is plotted using points, but those dense points appear to be several curves. To make it clearer, we zoom into smaller time windows, using points in Figure 6(b) and linespoints in Figure 6(c).

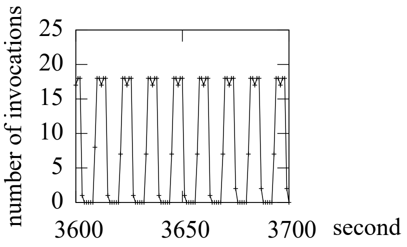


Fig. 7: Computed number of invocations that send messages.

messages, in every 1-second window. Figure 7 shows a similar zig-zag pattern as in Figure 6(c), which explains such a pattern.

APPENDIX V: PROOF FOR THEOREM 1

This section proves Theorem 1. All line numbers in this section refer to lines in Algorithm 1 through 3. We say that a node *accepts* a Merkle root r if the node adds r to its `root_accepted` at either Line 37 or Line 41. A node may accept the same r multiple times. In the overlay network, we call a path as an *honest path* if it (including the starting and ending node) contains only honest nodes and good edges. The *honest distance* between two honest nodes A and B is the length of the shortest honest path between A and B. The proofs will use superscript to indicate variables on a give node — for example, $\texttt{top\_root}^C$ refers to the `top_root` in the algorithm running on node C.

Roadmap. The following is a roadmap for the proofs. Appendix V-A presents several lemmas and then Theorem 8, which shows that the first phase enables the honest nodes to agree on the Merkle root. Appendix V-B eventually gives Theorem 12, which captures the agreement property of the second phase (for the fragments). Finally, Appendix V-C proves Theorem 1, by using Theorem 8 and 12.

A. Agreement on the Merkle Root

The proofs of all the lemmas/theorem in this section are deferred to [15]. Lemma 5 next roughly says that if an honest committee member A accepts a certain Merkle root r_0, then all other honest nodes must also accept r_0 within some rounds after that, assuming the algorithm has not already terminated by then. But there will be an exception — an honest node may accept two different roots r_1 and r_2, without accepting r_0.

Lemma 5. *Consider any honest committee member A and any honest node D, and let $g \in [0, d]$ be the honest distance between A and D. If A accepts r_0 in round i and if $i + g \leq 2dm + s - 1$, then by round $i + g$, node D must satisfy either one or both of the following properties:*

- *D accepts r_0.*
- *D accepts two different roots.*

Lemma 6 and 7 next intend to eventually show that if an honest non-committee member A accepts a certain Merkle root r_0, then all honest committee members must also accept r_0 within some rounds after that. Same as in Lemma 5, there will be an exception — namely, accepting two different roots r_1 and r_2 instead of r_0.

Lemma 6. *Consider any honest node A and any honest committee member D, and let $g \in [0, d]$ be the honest distance between A and D. If at Line 44 of round i, node A makes a push with a score of at least g, then we must have $i + g \leq 2dm + s - 1$, and furthermore node D must satisfy either one or both of the following properties in round $i + g$:*

- *D accepts r_0, where r_0 is the root contained in A's push.*
- *D accepts two different roots.*

Lemma 7. *Consider any honest non-committee member A and any honest committee member D, and let $g \in [1, d]$ be the honest distance between A and D. If A accepts r_0 in round i, then we must have $i+g \leq 2dm+s-1$, and furthermore node D must satisfy either one or both of the following properties in round $i + g$:*

- *D accepts r_0.*
- *D accepts two different roots.*

Intuitively, Theorem 8 next implies that exactly one of the following cases must happen at Line 28:

- All honest nodes have the same singleton set as the value for `root_accepted`; or
- $|\texttt{root\_accepted}| \neq 1$ on all honest nodes. (In this case, all honest nodes will eventually output $\perp$.)

Theorem 8 (Agreement on Merkle Root)**.** *Consider any execution of Algorithm 1, where at least one honest node has $|\texttt{root\_accepted}| = 1$ at Line 28. Then in this execution, all* *honest nodes must have the same* `root_accepted` *value at Line 28.*

B. Agreement on Fragments

The proofs of all the lemmas/theorem in this section are deferred to [15]. Recall from Section IV-C that in the second phase, a node B uses the Merkle root contained in its most promising push done so far in the first phase, as B's current guess for the final accepted Merkle root. Lemma 9 below says that under certain conditions, after an honest node A accepts a Merkle root, within a certain number of rounds, the guesses made by other honest nodes will become correct.

Lemma 9. *Consider any given execution of Algorithm 1, where* $|\texttt{root\_accepted}| = 1$ *at Line 28 on some honest node* A*. Let round* i *be when* A *first accepts the sole element* r_0 *in* $\texttt{root\_accepted}^A$*. Let* B *be any honest node (*B *can be* A *itself), and let* $g \in [0, d]$ *be the honest distance between* A *and* B*. Then in round* $i + g$ *and all later rounds, the push* p *(i.e., the most promising push) chosen by node* B *at Line 50 must contain* r_0*, if either of the following two conditions is satisfied:*

- *A is a committee member.*
- *A is a non-committee member and there exists some honest committee member D such that the honest distance between B and D is no more than $d - g$.*

Lemma 10 and 11 next reason about the agreement properties for the fragments, under certain conditions.

Lemma 10. *Consider any given execution of Algorithm 1, where at least one honest node has* $|\texttt{root\_accepted}| = 1$ *at Line 28. If there exists some honest committee member having* `frag_accepted` $=$ `true` *at Line 28, then all honest nodes must have* `frag_accepted` *being true at Line 28.*

Lemma 11. *Consider any given execution of Algorithm 1, where at least one honest node has* $|\texttt{root\_accepted}| = 1$ *at Line 28. If there exists some honest non-committee member having* `frag_accepted` $=$ `true` *at Line 28, then all honest committee members must have* `frag_accepted` *being true at Line 28.*

Building upon Lemma 10 and 11, Theorem 12 next shows that all honest nodes must agree on whether they accept the fragements:

Theorem 12 (Agreement on Fragments)**.** *Consider any execution of Algorithm 1, where at least one honest node has* $|\texttt{root\_accepted}| = 1$ *at Line 28. Then in this execution, all honest nodes must have the same* `frag_accepted` *value at Line 28.*

C. Proving Theorem 1

Theorem 1 (Restated)**. [guarantees of** OVERLAYBB**]** *In Algorithm 1, if the committee has at least one honest member, then*

- *All honest nodes must return the same object.*
- *If the broadcaster is honest, then all honest nodes must return the object broadcast by the broadcaster.*

Finally, regardless of the committee, Algorithm 1 always returns within $2dm + s$ *rounds.*

Proof. We first prove that all honest nodes must return the same object. If all honest nodes output $\perp$, we are done. Otherwise some honest node must satisfy Line 28 with $|\texttt{root\_accepted}| = \{\texttt{r}_0\}$ and $\texttt{frag\_accpeted} = \texttt{true}$, for some r_0. Let A be any honest node. Then by Theorem 8 and Theorem 12, node A must also have $\texttt{root\_accepted}^A = \{r_0\}$ and $\texttt{frag\_accepted}^A = \texttt{true}$. Next, it suffices to prove that $\texttt{all\_frag}^A$ contains all s fragments corresponding to r_0. Let t_0 be the round during which A first set $\texttt{frag\_accepted}^A$ to be true. In round t_0, since Line 60 or Line 64 must be satisfied on A, we must have $t^A_{\text{frag}} \neq \infty$ and $t^A_{\text{root}} \neq \infty$. This means that t^A_{root} has already been assigned some value in or before round t_0. Since t^A_{root} is never assigned a value larger than the current round, we must have $t^A_{\text{root}} \leq t_0$. Then by Lemma 9, in round t_0 the most promising push chosen by A at Line 50 must contain r_0. Since A later sets $\texttt{frag\_accepted}^A$ to be true in that round, by Line 58, all the s fragments corresponding to r_0 must already be in $\texttt{all\_frag}^A$.

We next prove that if an honest broadcaster A broadcasts an object O, then all honest nodes must return O. Given we have already proved that all honest nodes must return the same object, it suffices to show that A will return O. Let the Merkle root for O be r_0. By Line 20, no other Merkle roots will ever be processed by any honest node, since they do not have a signature from A. In round 0, node A must reach and satisfy Line 35. Then A will add r_0 to `root_accepted` and set t_{root} to be 0. Since no other root will ever be processed by A, A must have $|\texttt{root\_accepted}| = 1$ at Line 28. Next, one can trivially follow the steps in Algorithm 3 and verify that during round $s - 1$, node A must have $t_{\text{root}} = 0$ and $t_{\text{frag}} = s - 1$ at Line 60. Hence node A must later set `frag_accepted` to be true. Finally, one can trivially verify that at Line 28, node A must have all s fragments corresponding to r_0 in `all_frag`. Putting everything together, A must return O.

Finally, it is obvious from the pseudo-code that the algorithm always returns within $2dm + s$ rounds, regardless of whether the committee has any honest member. $\square$