

An Empirical Study of Consensus Protocols’ DoS Resilience

Giacomo Giuliari
ETH Zurich
Mysten Labs

Alberto Sonnino
Mysten Labs
University College London (UCL)

Marc Frei
ETH Zurich

Fabio Streun
Anapaya Systems

Lefteris Kokoris-Kogias
Mysten Labs
IST Austria

Adrian Perrig
ETH Zurich

Abstract

With the proliferation of blockchain technology in high-value sectors, consensus protocols are becoming critical infrastructures. The rapid innovation cycle in Byzantine fault tolerant (BFT) consensus protocols has culminated in HotStuff, which provides linear message complexity in the partially synchronous setting. To achieve this, HotStuff leverages a leader that collects, aggregates, and broadcasts the messages of other validators. This paper analyzes the security implications of such approaches in practice, from the perspective of liveness and availability.

By implementing attacks in a globally-distributed testbed, we demonstrate that state-of-the-art leader-based protocols are vulnerable to denial-of-service (DoS) attacks on the leader. Our attacks, demonstrated on committees of up to 64 validators, manage to disrupt liveness within seconds, using only a few tens of Mbps of attack bandwidth per validator. Crucially, the cost and effectiveness of the attacks are *independent* of the committee size. Based on the outcome of these experiments, we then propose and test effective mitigations. Our findings show that advancements in both protocol design and network-layer defenses can greatly improve the practical resilience of BFT consensus protocols.

CCS Concepts

• **Security and privacy** → Denial-of-service attacks; Distributed systems security; • **Computer systems organization** → *Dependable and fault-tolerant systems and networks*.

Keywords

DDoS Attacks, Consensus Protocols, Blockchain

1 Introduction

For more than four decades, researchers have been studying Byzantine fault tolerant (BFT) consensus protocols [16, 18, 27] in order to facilitate the development of dependable distributed systems. As blockchains have grown in popularity, there has been an increasing interest in developing high-performance consensus systems, with early studies proposing committee-based protocols to improve over Bitcoin’s [54] throughput of 7 transactions per second. BFT consensus protocols have since been shown to increase blockchain throughput and reduce latency [9, 44], and they are rapidly becoming the standard in proof-of-stake architectures [7, 25, 46]. These efforts resulted in the creation of HotStuff [67] and its followup works [1, 35], which have linear message complexity in the partially-synchronous setting: They achieve this by electing a *leader*, selected

among the validators in the committee, that is responsible for collecting, aggregating, and broadcasting the messages generated by other validators.

Although this leader-centric strategy greatly reduces the overall communication complexity—which ultimately results in improved theoretical scalability—it also exposes the protocol to liveness attacks. All that is required for a protocol round to be unsuccessful is for the leader to lose synchronization with the rest of the system. This challenge is compounded by the need for blockchain protocols to offer censorship resistance, which is now addressed by several proposals via the use of alternating leaders.

We show that this weakness can be exploited in practice by launching denial-of-service (DoS) attacks on HotStuff. To this end, we set up a large and globally-distributed testbed of validators running the reference HotStuff implementation, and perform hundreds of experiments with multiple attack vectors, and covering a broad range of threat models. Our results unambiguously show that a few Mbps of attack traffic suffice to halt the consensus and compromise liveness within seconds, even when the adversary is external to the committee—i.e., there are no Byzantine validators. Most importantly, the presence of the leader makes the cost and effectiveness of these attacks *independent* of the committee size, as the adversary can easily track and target each subsequent leader.

Our attacks can be interpreted in several ways. One is that HotStuff—and specifically the implementation we use in the experiments—can be attacked. However, this is not the goal of this paper. Instead, our intent is to investigate the practical resilience of consensus protocols to DoS attacks, and to evaluate general solutions to protect consensus protocols in adversarial environments—such as a high-stakes blockchain deployed on the public Internet. Although the space of possible mitigations is extremely large, there are two immediate avenues for increasing the survivability of consensus protocols: (i) to explore more resilient protocol designs, and (ii) to integrate network-layer defenses to protect validator-to-validator communication from DoS. Therefore, we first consider fully-asynchronous consensus protocols, which aim to operate under extremely unfavorable network conditions. Then, we turn to the network security literature to identify DoS defenses that are amenable to the decentralized consensus setting.

Asynchronous consensus protocols—albeit designed to have better security and availability—have seen little attention by the distributed consensus community as they are often stigmatized as unrealistic, slow, or hard to implement. Nevertheless, we ask the question of whether they can withstand our attacks better than leader-based protocols. We therefore execute another set of attacks against Tusk [24], a recently proposed asynchronous consensus

protocol that promises speed and simplicity. We deploy the existing Tusk codebase, which is considered for adoption in three blockchain startups [14, 21, 47]. The results are encouraging, showing that Tusk is more robust than HotStuff. Its progress cannot be trivially paralyzed, thus indicating improved robustness of asynchronous consensus protocols. However, the transaction throughput of the protocol decreases under attack, and targeting more than f validators in a committee of $3f + 1$ breaks the liveness threshold and, as expected, the protocol halts.

To conclude our exploration on the *practical* resilience of state-of-the-art consensus protocols, we investigate promising network-based defenses to further protect validators. Unfortunately, we have to discard most traditional approaches, such as Cloud-based DoS filtering, which introduces too much centralization for a viable blockchain network, or VPNs, which potentially introduce new attack vectors [63]. We finally test the effectiveness of traffic authentication and rate-limiting (ARL) as a DoS protection mechanism. We implement an ARL prototype where source authentication is based on symmetric-key message-authentication codes (MACs), yielding a system with minimal computational requirements. Thus, our prototype can be transparently deployed at the validators, does not require changes to the consensus, and does not introduce any consensus delay. When active, ARL blocks all our attacks, and consensus proceeds unscathed.

With this paper, we then contribute to the **study of the attack resilience of consensus protocols in their integration within real networks**, and beyond their theoretical guarantees. We find that leader-based consensus protocols such as HotStuff, although in theory resilient to up to f failures, can fall prey to attacks on liveness because of their reliance on a single leader. Leader-less asynchronous consensus protocols such as Tusk, on the other hand, show better resilience. Finally, we argue that full DoS resistance is only achieved when considering further defenses at the network layer. We thus hope to highlight how the different synchrony models—under which the security properties of consensus protocols have been proven—hardly match the reality of today’s Internet, where traffic can be dropped, rerouted, or spoofed. Therefore, we suggest that safe and available distributed consensus requires a holistic approach, reconciling abstract models with real deployments.

2 Background

This paper focuses on quorum-based consensus protocols. These protocols have a well-known set of participants—called *committee*—a subset of which (typically $< \frac{1}{3}$) can be faulty or adversarial, also called *Byzantine*. In the protocols we consider, members of the committee—the validators—authenticate each other using cryptographic signatures.

Additionally, we focus on partially synchronous [29] and asynchronous [16] protocols that can be deployed over an unreliable network, such as the Internet. Partially synchronous protocols typically feature lower latency than asynchronous protocols when the network is reliable, and work by optimistically electing a leader to drive the protocol and rotating it regularly. Such protocols suffer larger performance degradation during periods of asynchrony or when the leader is Byzantine. On the other hand, asynchronous protocols do not rely on a leader to drive the protocol and are therefore

more resilient. However, they have higher latency than partially synchronous protocols, as they need to introduce randomization in the execution to withstand network asynchrony [34].

Specifically, we analyze two state-of-the-art consensus protocols: (i) HotStuff [67] as an example of partially-synchronous consensus, and (ii) Tusk [24] as an example of asynchronous consensus. We chose these particular protocols for several reasons. First, both protocols feature open-source implementations with well-documented benchmarking scripts to measure performance under a variety of conditions. Second, their implementations are comparable: They are both written in Rust, use similar libraries (network, storage, and cryptography), and are built according to similar design choices, thus allowing for a fair comparison. Finally, HotStuff is used at the core of many open-source projects and companies, such as Celo, Cypherium, Flow, Monad, and Diem.¹ This paper thus offers insights on possible threats to deployed systems.

The following background on consensus protocols is needed to understand the attacks presented in this paper.

2.1 HotStuff: Partially-Sync. Consensus

HotStuff [67] operates in a round-by-round manner, electing a leader in each round among the committee to balance validator participation.

The leader proposes an extension to the longest chain of requests that it already knows;² the rest of the validators then vote for the extension, unless the extension conflicts with a longer chain they know. Validators finally send their votes to the next leader to help them learn the longest safe chain. If $2f + 1$ validators send votes to the next leader in a timely manner, that leader can gather them in a data structure called the *quorum certificate* (QC), and build a new block (the next proposal). If there are three consecutive blocks in the chain, B_k, B_{k+1}, B_{k+2} , which are proposed in consecutive rounds, and each block has a QC, then the protocol has reached consensus on block B_k ; all honest validators eventually commit B_k . Additionally, validators maintain a timer to track progress and preserve liveness despite faulty leaders. When the timer expires and a validator has still not received a proposal, it broadcasts a timeout vote. A validator gathering enough timeout votes can form a timeout certificate (TC) and advance its round. Every time a round fails, timeout periods are increased, allowing lagging validators to catch up and enabling the protocol to commit eventually.

2.2 Tusk: Asynchronous Consensus

The Tusk [24] consensus protocol provides safety and liveness even in asynchrony, and does not make direct use of a leader to drive consensus. We, therefore, use Tusk as a baseline to experimentally test the robustness of *leader-less* asynchronous consensus in comparison to the leader-based, partially-synchronous HotStuff.

Tusk is split into two sub-protocols, (i) a data dissemination system called Narwhal, which forms a directed acyclic graph (DAG) of batches of transactions, and (ii) a decision rule on how to totally order the vertices of the DAG and reach consensus.

¹celo.org, cypherium.io, flow.com, monad.xyz, diem.com

²Usually, leaders collect batches of requests to propose, referred to as blocks. Hence, the HotStuff protocol forms a chain of blocks (or a blockchain).

Narwhal: Building the DAG. Narwhal proceeds in rounds to build a DAG on batch metadata. Data dissemination is symmetric among validators, and therefore it does not have a single point of failure, such as a leader. In each round, each validator prepares and sends to all other validators—via consistent broadcast [17]—a message with the batch metadata corresponding to a DAG vertex. This metadata contains the batch digest and $2f + 1$ references to vertices from the previous round. Upon receiving such a message, a validator replies with a signature iff (i) It has already stored the data corresponding to the digests in the vertex (for data-availability); and (ii) it has not replied to this validator in this round before (for non-equivocation). The sender forms a quorum certificate from $2f + 1$ such signatures and sends it back to the validators as part of its vertex for this round. A validator advances to the next round once it receives $2f + 1$ vertices with valid certificates.

Tusk: Interpreting the DAG. Tusk takes the causally ordered DAG constructed by Narwhal and totally orders its blocks. While the details of Tusk’s operation are beyond the scope of this paper, it is important to notice that Tusk achieves the total ordering of transactions with zero extra communication and modest computation. That is, every validator determines this total block order only based on its view of the DAG and some shared randomness—a distributed perfect random coin [49]—derived from the blocks.

Consequently, DAG ordering is light-weight and has a reduced attack surface compared to the Narwhal subsystem, which instead requires exchanging and processing network messages, and verifying the signatures contained within.

2.3 Denial of Service

Denial of service (DoS) is an umbrella term for a broad variety of attacks against the availability of interconnected services. The prevalent form of DoS attack today is the volumetric distributed-denial-of-service (DDoS) attack, whereby an adversary—the *bot-master*—directs the traffic of thousands to millions of compromised Internet hosts—the *bots*—towards a target endpoint. The resulting traffic flood, which can reach multiple Tbps, depletes the computation, memory, or bandwidth resources of the targets, forcing packet drops and preventing legitimate traffic from reaching the service. Attacks can target all layers in the network stack, e.g., by congesting network links (network layer), by exhausting state with a huge number of open connections (transport layer), or by draining compute power on the host with resource-intensive requests (application layer). Some of the most recent, high-profile targets include governments³ and large organizations.⁴

In this paper, we study DoS attacks targeting validators running a consensus protocol. In particular, we exploit the reliance of such protocols on cryptographic signatures to authenticate all consensus messages: We design and implement *signature flooding attacks* where validators are forced to verify tens of thousands of signatures per second. Since signature verification is a compute-intensive operation, a relatively low-rate attack—compared to purely volumetric floods—can overwhelm a validator.

In §4, we provide a more formal discussion of the threat models and attack vectors, while in §7, we discuss the challenges that the distributed nature of consensus protocols poses to DoS defenses.

3 Methodology

We present a general framework to evaluate the resilience of consensus protocols to practical DoS attacks. We first define the threat model, then provide a stepwise experimentation procedure, and finally describe novel metrics to evaluate DoS attack resilience in consensus protocols.

3.1 Threat Model

Given a consensus protocol running on a wide-area network—typically the Internet—our model considers two types of adversaries, of increasing power.

The *external* adversary does not control compromised validators in the consensus committee, and, therefore, cannot authenticate protocol messages. However, such an adversary can impersonate clients and submit transactions to the consensus, or send adversarial traffic from other hosts in the network (the “botnet”).

The *internal* adversary controls up to f Byzantine validators among the committee of $3f + 1$ validators. In contrast to the external adversary, the internal adversary has access to valid key material and can authenticate protocol messages. Crucially, this is the most powerful adversary that does not breach the theoretical requirements for the security of the consensus protocol.

In both cases, we consider the adversary to be in control of a botnet, with a large number range of IP addresses from which it can send traffic. Further, the adversary can observe the current state of the protocol, and specifically the current round number. This assumption is realistic, as most consensus protocols do not attempt to hide their protocol messages since they can easily be guessed by observing the blockchain, and are often needed by light clients as proofs of commits. Therefore, any entity can obtain the round number by (i) passively observing the traffic of validators; or (ii) through publicly available information; or even (iii) by observing traffic patterns—e.g., the leader is the validator that broadcast messages to all other validators. Internal adversaries have even more insight into the state of the protocol as they are directly participating with a subset of Byzantine validators.

Finally, the adversary model of the BFT protocols under consideration also assumes that the adversary has full control over network communications. In particular, the adversary may spoof the source IP address, either by using on-path bots, or by hijacking the IP address of validators. This assumption is *not* required for the attacks we present in the paper to succeed, but will aid in the analysis of robust mitigations.

3.2 DoS-Resilience Evaluation Pipeline

We propose the following experimental procedure in three steps for the practical evaluation of the DoS resilience of consensus protocols.

#1: Static-leader attacks. First, evaluate the resilience of a single validator to DoS attacks to establish a baseline, and reveal implementation aspects that may facilitate attacks. The results also serve as an indicator of the required resources for an attack on multiple

³E.g., the May 2021 attacks on the Belgian parliament [20].

⁴See Cloudflare’s list of famous DoS attacks [22].

validators. In the context of leader-based consensus protocols, we are mainly interested in the resilience of the leader to DoS attacks.

#2: Fixed-subset attacks. Second, test against an adversary targeting a fixed set of validators at the same time. Even if only a small set of validators are targeted, such attacks can significantly slow down the protocol, potentially rendering it practically unusable. This second step is also useful to gauge the relevance of the theoretical guarantees the protocol offers. In principle, all BFT consensus protocols should still be live with up to f validators under attack.

#3: Leader-tracking attacks. Consensus protocols typically assume a synchronous, partially-synchronous, or asynchronous network model, with up to f fixed Byzantine (or faulty) validators. None of these models restricts the attack capabilities of clients or external adversaries. In an Internet setting, then, it is reasonable to assume that the adversary can rapidly change targets during an attack. We therefore propose to study this case separately in our methodology.

A leader-tracking attack concentrates the adversary’s resources against the leader, trying to emulate the effects of a static-leader attack in the more realistic dynamic setting. To succeed, the adversary has to overcome the additional difficulty posed by the quick rotation of leaders (e.g., ≈ 5 rounds/s in HotStuff). In practice, the adversary may not have enough time to crash the leader before a new one is elected.

Finally, a number of recent works propose to increase the resilience of leader-based BFT consensus protocols by introducing unpredictability in the leader selection procedure [11, 15, 28, 37, 51]. In these protocols, the committee elects each leader randomly at the beginning of the round, e.g., by using an unpredictable shared random coin, making targeting the leader harder, or even infeasible. The evaluation of leader-tracking attacks needs to also consider such unpredictable leader election sequences.

3.3 Evaluation Metrics

We are interested in attacks which significantly degrade or completely disrupt a protocol’s performance with the least cost or effort for an adversary. The effort is measured in terms of resources the adversary must invest to succeed in the attacks, such as the amount of attack traffic (bandwidth utilization) or the number of attack machines (computational power).

To evaluate the effectiveness and the cost of attacks, we define the following metrics, and illustrate them in Fig. 1 with an example taken from the attacks on HotStuff.

Normalized Commit Rate under Attack. Attacks on liveness congest the validators such that they take more time to commit new blocks. A maximally effective attack sufficiently delays the commit to reach a timeout and force a view change. In this case, no blocks can be committed, and the consensus is effectively halted. We capture this effect by measuring the commit rate—the number of commits per second—after the attack start, and compare it to the commit rate in steady-state operation. Figure 1 shows that before the attack, HotStuff progresses with roughly 5 commits per second (cmt/s), while after the attack starts the rate drops to zero. We express this metric as the ratio of the *commit rate after the attack start* to the *commit rate without attacks*. The smaller this number, called *normalized commit rate*, the more effective the attack.

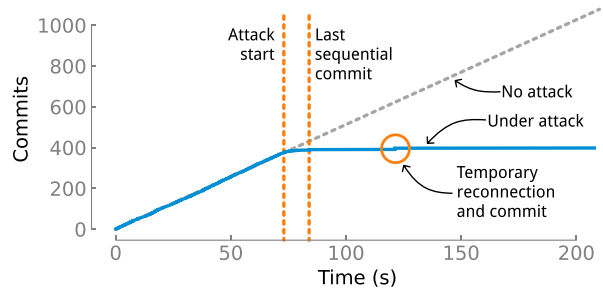


Figure 1: Example HotStuff run with/without attacks. A committee of 64 validators is under attack from 8 adversary machines (no Byzantine validators). The gray dotted line represents the unperturbed progress of the protocol.

Time-to-Last-Commit. In case the attack is successful in completely halting consensus, the normalized commit rate is zero. This metric therefore cannot capture the difference between distinct, but successful, attacks. We then introduce the time-to-last-commit (TLC) to discern the effectiveness of attacks in these cases. The TLC measures the time elapsed from the instant the first adversary machine starts attacking the consensus to the time of the last commit. Since the protocol may temporarily re-synchronize and commit a block, we compute the TLC as the time after which there are no commits for 30 consecutive seconds (these events are nevertheless extremely rare in our experiments). In Fig. 1, the TLC is represented by the distance between the vertical bars of denoting attack start and last sequential commit. More powerful attacks halt the consensus faster, and therefore have lower TLCs.

Committee and Adversary Size. In principle, a larger committee should be able to resist more powerful adversaries, controlling more computational and network resources. Therefore, we also investigate the resilience of the committee relative to the size of the adversary.

4 Attacking HotStuff

By applying the methodology described in §3, we show that attacks on the liveness of leader-based consensus protocols—here exemplified by HotStuff—are practical. Starting from an analysis of the space of threat models and attack vectors, we devise a set of proof-of-concept attacks and experimentally prove that they can rapidly, inexpensively, and *indefinitely* prevent liveness.

4.1 Attack Vectors

The decentralized, trust-minimizing nature of the HotStuff consensus protocols enables multiple avenues for adversaries trying to delay or halt the protocol by flooding validators with packets. We summarize the three categories.

Client Traffic. The adversary impersonates a large number of clients, and overloads the validators with legitimate transaction messages. This attack aims at exhausting the computational resources of the validators, by forcing the processing of many transactions. Previous work has shown that increasing the number of

client transaction requests can drastically increase consensus delay [24, 67] up to the point of halting consensus [62]. Nevertheless, a flood of client traffic is in general less problematic, as the focus of protocols is typically consensus liveness. Rate-limiting the number of requests from clients is a traditional defense that protects the consensus core against this attack.

Our experiments confirm that this DoS effect is present in the HotStuff implementation under test. However, since the countermeasure is easily deployable, we do not exploit client traffic as an attack vector in this paper.

Unauthenticated Protocol Traffic. In blockchain applications, validators can leave and join frequently. Therefore, validators have to process messages from possibly any Internet host. The validators’ identity is only tied to the knowledge of a public/private key-pair, and thus all messages are authenticated with signatures. Then, even *external* adversaries can initiate *signature-flooding attacks*, where they target honest validators with protocol messages containing bogus signatures. Even if the verification step fails—an external adversary does not have a valid private key—the computational overhead of verifying a flood of signatures may still overwhelm a validator and cause it to drop legitimate consensus messages.

Authenticated Protocol Traffic. Internal adversaries can create valid protocol messages and therefore, even if they cannot compromise safety, they have many opportunities to create computation and memory overhead on other validators. Signature floods are particularly threatening because the signatures will pass the authentication checks, and the messages will proceed to create additional overhead.

4.2 Evaluation Setup

HotStuff Implementation. In our attack experiments, we use the reference HotStuff implementation in Rust [61]. This instantiation uses the 2-chain version of the protocol [35] and implements the DiemBFT pacemaker [10]. It is multi-threaded and uses tokio⁵ for asynchronous networking, ed25519-dalek⁶ for elliptic curve based signatures, and data-structures are persisted using RocksDB.⁷ It uses TCP to achieve reliable point-to-point channels, necessary to correctly implement the distributed system abstractions. Every message is authenticated with a signature over the message’s digest, i.e., the 32-Byte output of a cryptographic hash function. We set the target transaction rate relatively low (10 000 tx/s), to avoid overloading the validators and thus facilitate the observation of the results of our attacks.

Adversary Implementation. The adversary is also implemented in Rust, using the tokio library, and reusing the message definitions of the HotStuff implementation.

Committee and Adversary Deployment. For our experiments, we deploy a network of HotStuff validators on a global wide-area network composed of AWS virtual private servers (VPSes). These VPSes (AWS m5d. 2xlarge) have 8 virtual cores, 32 GB of memory, and up to 10 Gbps of available network bandwidth, providing a good tradeoff between performance and cost. Depending on the experiment, 8 to 64 validators are uniformly spread across 4 AWS

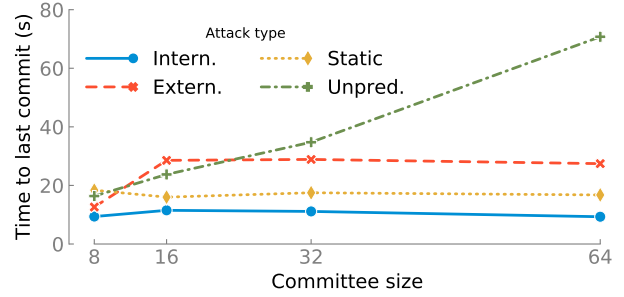


Figure 2: Average TLC of leader-tracking attacks on HotStuff. Showing leader-tracking attacks with internal —●— and external -■- adversaries; static-leader attacks -◇-; and leader-tracking attacks with an unpredictable leader-election schedule -+. See Figs. 7 and 8 in Appendix A for further analysis.

data centers spanning the globe.⁸ Our experiments thus run on a validator network of realistic size and communication latency. The adversary’s deployment is similar, in that it uses the same number and type of VPSes, and data center distribution. We can thus compare the computational power of the adversary and the committee to obtain a coarse estimate of the required attack power relative to the committee size.

Evaluation Orchestration. An orchestration script is responsible for automatically creating the VPSes on AWS, deploying and building the code, initiating the consensus protocol and the attacks, and collecting the run logs. In each run, after the committee is started, we wait for it to synchronize and reach steady-state operation. After 60 seconds have passed, we start the adversary’s machines. The experimental pipeline comprising multiple attack runs with different parameters, log parsing, and plotting, is implemented with the Snakemake workflow-management tool [53].

5 HotStuff Attack Results

5.1 Static-Leader Attacks

We evaluate the static-leader attack on 16 different combinations of committee sizes (in {8, 16, 32, 64}) and adversary sizes (similarly, {8, 16, 32, 64}). For each committee size and number of adversaries, we run the attacks twice (32 attack runs in total) and for 180 seconds. We modify the leader-election module of the HotStuff validators to always elect the same leader—instead of proceeding in a round-robin fashion—so that the adversary may easily target the leader. The adversary is external to the committee, and targets the validators with a Vote signature flood, carried out as follows.

The Vote Signature Flood. The machines in the adversary’s botnet send as many Vote messages as possible to the validator. These messages are crafted to force a signature check on the validator.

First, the author field is set to the public key of one of the committee members, impersonating its identity. Thus, the message passes the initial check at the target, which ensures that all Vote messages contain the identity (public key) of another known committee member. Then, the round number for which the vote is valid, contained

⁵<https://tokio.rs>

⁶<https://github.com/dalek-cryptography/ed25519-dalek>

⁷<https://rocksdb.org>

⁸us-east-1, eu-north-1, us-west-1, ap-northeast-1

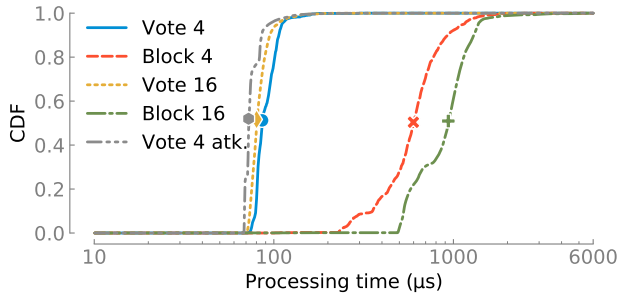


Figure 3: Vote and Block processing time distributions. The values indicate the committee size. Vote processing time is almost constant, taking 70 – 80µs in the median, independently of committee size and attacks. Block processing is more expensive (500 – 1000µs), and depends on the committee size.

in the round field, is set to a high value such that it will always be greater than the actual round number during the protocol execution. The message then passes a second check that requires the vote to be for the current or a *future* round. From the consensus standpoint, allowing future votes to be processed is beneficial, as it may help a lagging validator in forming the quorum for a view change and move to the latest round⁹. However, accepting future votes also lowers the synchronization requirements on the external adversary, who does not need to precisely know the round number. Finally, the digest—containing the hash of the block for which the vote is—and the signature are filled with random bytes.

Results. Results are indicated by the “Static” ---◆ line in Fig. 2 (and magnified in Fig. 7a in the Appendix). We find that attacks against a static leader are highly effective, and consensus is halted within 10–25 seconds—16 seconds in the median—independent of the committee size and number of adversaries. A further breakdown of the TLC behavior depending on the number of adversaries machine can be seen in Fig. 7a in the Appendix.

This experiment confirms that the processing overhead of the Vote signatures forces the target machine to drop packets, among which are the valid votes from the committee. Since the attack targets the (static) leader of the committee, it is able to rapidly compromise liveness. As a single Vote message requires around 80 µs in the median (Fig. 3) to authenticate and then discard, a single core can at most process 12 500 votes per second. Once the adversary is able to deliver votes at a rate above this threshold, the validator starts to buffer votes, and eventually has to drop packets. We monitor the number of drops on the network interface card (NIC) with the `nstat` tool, and confirm that the number of dropped packets sharply increases on the validator under attack.

5.2 Fixed-Subset Attacks

Having established the resistance of a single validator to attacks, we then investigate fixed-subset attacks to test the consensus protocol’s resilience to up to f failures. In principle, either the adversary is able

⁹A straw-man improvement to the protocol is requiring voters to attach a QC to the vote message, proving that a quorum of validators reached the voted round. This, however, forces the leader to check an extra QC per vote.

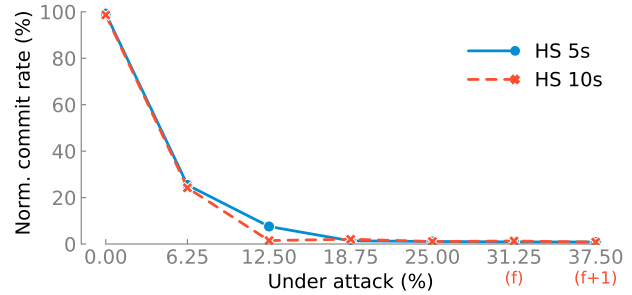


Figure 4: Fixed-subset attack. The adversary targets 0 to 6 validators in a 16-validator committee ($f = 5$). HotStuff (HS) is here set up with two timeout settings (5s and 10s).

to crash more than f validators, or the consensus should be able to make progress. In this experiment, t validators in a committee of 16 ($f = 5$) are targeted by an adversary, with t in $\{1, \dots, 6\}$. The adversary is *external* to the committee, and employs the Vote signature flood against target validators. We then measure the normalized commit rate across different runs, which is expected to be close to zero only for $t = f + 1 = 6$. We further use two different timeout settings (5 and 10 seconds), to test whether this parameter improves the protocol’s resilience. The common expectation is that with longer timeout duration the chance that valid messages manage to reach the validators under attack increases, which in turn should increase the chance to achieve a commit. For each number of targets t , we run an attack twice for each timeout setting, for a total of 24 experiments. Each experiment runs for 10 minutes, and the adversary is started after 60 seconds of consensus warm-up time. We thus measure the normalized commit rate for 9 minutes to capture the long-term effects of the attack.

Results. The lines *HS5s* —● and *HS10s* -◆ in Figure 4 show the result of the experiment runs. Somewhat surprisingly, we find that the consensus is halted indefinitely with just $t = 3$ validators out of 16 under attack (18.75%).

The reason for this rapid decrease in the normalized commit rate is that, for each leader under attack, the protocol is halted for a full timeout of 5 or 10s. The effect is further worsened by a performance optimization of the HotStuff protocol. In the implementation we target, to commit the leader needs to perform three rounds of broadcast communication to all other validators, and collect their replies. Thus, these communication rounds can be pipelined and executed in parallel by leaders of consecutive rounds, increasing the transaction throughput of the consensus. The “2-chain” pipelining still implies that to commit, *two consecutive leaders must be live and honest*, which severely weakens the protocol. Disrupting a single validator then causes not one but two consecutive rounds to fail, and consequently f crashed validators cause $2f$ failed rounds, reducing the commit rate even more. Therefore, assuming a base commit rate of 5 cmt/s, if e.g., 3 leaders crash in a committee of 16, the protocol is blocked for $3 \cdot 5s + 3 \cdot 0.2s = 15.6s$, versus the $10 \cdot 0.2s = 2s$ in which it can make progress. Equivalently, without an attack HotStuff would have committed almost 6× more blocks.

5.3 Leader-Tracking Attacks

In this attack scenario, the adversary tries to continuously DoS the current leader to disrupt liveness. We present the results for both external and internal adversaries.

Leader-Tracking Implementation. As discussed in §3.1, the adversary has many ways to learn the round number, even if it is external to the committee. We abstract away the details of the concrete way in which the adversary obtains this information, and simply add a subroutine on the validators that, upon view change, broadcasts the current round number to the adversary machines. How the adversary monitors the network in practice is orthogonal to our attack.



Attack Mechanism and Parametrization. Knowing the latest up-to-date round number, the adversary can identify the current leader as well as the upcoming leaders in the sequence (we later relax this assumption with the *unpredictable leader experiments*). The botnet can thus target the current leader's machine and a number of validators that will become leaders in the following rounds. In the following experiments, the total number of validators under attack t is chosen as $t \in \{2, 8\}$: $t = 2$ is always less than $f + 1$ for committee sizes $\{8, 16, 32, 64\}$; $t = 8$ amounts to an $> f + 1$ attack when the committee size is either 8 or 16. In this way, we can explore the effect of t in larger committees, while comparing it to an $f + 1$ attack in smaller ones.

The external adversary uses the same Vote-flood attack vector presented in the previous attacks. For the internal adversary, we introduce the Block signature flood.



The Block Signature Flood. Similarly to the Vote signature flood, the adversary sends a flood of Block protocol messages to the target validators, inducing computation overhead and eventually forcing dropped packets. The main difference between Vote and Block floods is that an internal adversary can leverage a valid private key to sign Block messages, and cause an even higher overhead.

The author field in the message is set to the identity of the validator controlled by the internal adversary. To pass the preliminary checks on the block's validity, the round field is set to a round number far in the future such that the validator is going to be the legitimate leader for that round. A signature is performed on the digest of the message.

Different to a Vote message, however, the Block message also contains a QC field with the quorum certificate for the previous block (see §2.1). The QC itself is an aggregate of the votes of at least $2f + 1$ validators for the previous block in the chain. After authenticating the signature—which succeeds because the internal adversary has a valid keypair—a validator processing the Block then checks that the QC is valid by verifying all signatures contained within. In this HotStuff implementation, the QC verification subroutine invokes a batched signature-verification function, which is more efficient than authenticating all signatures sequentially. However, this function only returns after the full batch has been processed. The adversary can then insert one (bogus) signature for each committee member, causing a higher overhead for each block depending on the committee size (Fig. 3).

Results. The TLC of the internal  and external  adversaries are shown in Fig. 2. As expected, the internal adversary has

the lowest TLC across all experiments, confirming the effectiveness of Block messages as attack vectors. The TLC for the external attack is slightly higher: 21 vs 10 seconds in the median. Most importantly, in both attacks the TLC is independent of the committee size. Further breakdowns are in Fig. 7 and Fig. 8 in the Appendix.

Comparing the static-leader attack  with the leader-tracking attack , we can see that even if they use the same attack vector—Vote flood by an external adversary—the latter has a slightly higher TLC. This is due to several factors. First, in the static-leader experiment the attack traffic is focused on a single machine; in the leader-tracking attack, traffic is spread across t validators, reducing the effect on any single validator. Second, the adversary can temporarily get out of sync with the rounds of the consensus. Thus, it may target validators that are not in the set of next leaders, yielding a less efficient attack. As shown in Fig. 8 in the Appendix, targeting more upcoming leaders produces a faster and more effective attack. In both cases however, the attack effectiveness is still independent of the committee size.

When is the adversary unsuccessful? Finally, we run an experiment to see when an adversary *may not have enough resources to be successful* to establish the minimum requirement for the botnet. To this end, we run the same external adversary as above against a committee of 64 validators, but now with only $\{1, 2, 4, 8\}$ adversary machines. The results are shown in Fig. 5: We see that 1 and 2 adversary machines are unable to generate enough attack traffic to completely disrupt consensus, although they still greatly reduce the commit rate. Each experiment lasts 180 s, with the attack starting after 60 s. Therefore, a TLC above 120 s—such as in the case of 1 and 2 adversary machines—indicates that the attack was not successful. However, they can still affect the commit rate compared to the non-adversarial case.

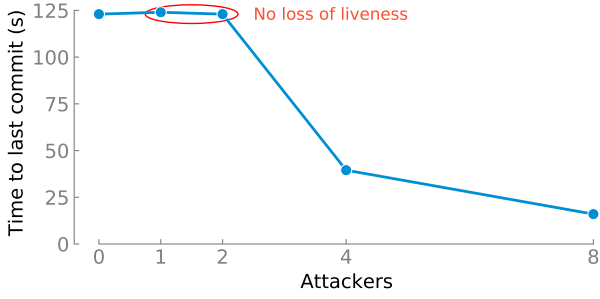
5.4 Unpredictable Leader-tracking Attacks

As shown above, the adversary can exploit the deterministic leader election sequence to target upcoming leaders and increase the effectiveness of the attack. However, certain consensus protocols (see §3.2) use an *unpredictable* leader election, where the committee elects the leader randomly at the beginning of each round. Therefore, leader-based attacks may become harder, or even infeasible.

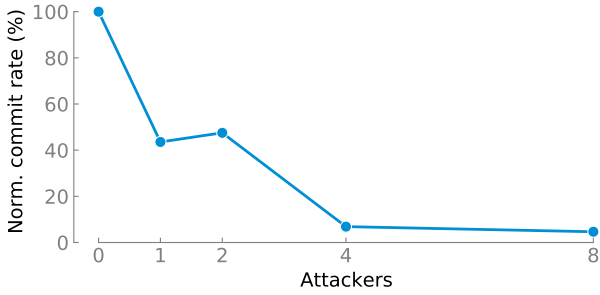
We thus complete our exploration of attack scenarios by implementing an adversary that does not know the leader election sequence, and learns the identity of the current leader either by (i) participating in the election with a compromised validator (internal adversary); or by (ii) monitoring the network and observing which validator sends Block proposals (external adversary).

Simulating Unpredictability. The HotStuff implementation used in the experiments does not provide an unpredictable leader election. Instead of implementing this functionality ourselves, we keep the round-robin leader election of the original implementation and instead simulate “unpredictability” on the adversary side.

To this end, we implement an adversary that only targets the leader for the current round, thus representing an adversary not knowing who the next leaders will be. The difference between internal and external adversaries then lies in the *time* at which they learn who the current leader is. Internal adversaries are part of the committee that elects the leader, and therefore learn the identity of



(a) TLC in the number of adversary machines.



(b) Normalized commit rate in the number of adversary machines.

Figure 5: Minimum requirements for an attack.

the new leader as soon as a view change occurs. External adversaries can only infer this information by observing the network traffic and the contents of Block messages, and thus learn the identity of the new leader only after the leader has proposed a Block.

Results. The TLC of the unpredictable leader-tracking attacks $\dashv\vdash$ is shown in Fig. 2. Each point in the plot is the average of 16 experiments ($\{8, 16, 32, 64 \text{ adversaries}\} \times \{\text{internal, external}\} \times \{2 \text{ reruns}\}$). The key insights of this experiment are that (i) the adversary can still paralyze the consensus by attacking the single leader in a few tens of seconds, but (ii) the TLC now appears to be linear in the committee size. This second observation can be explained by the fact that the adversary is always behind in its view of the consensus, and therefore cannot flood its targets before the legitimate votes from other validators reach the leader, and the consensus proceeds to the next round. However, even if the adversary is too late to stop the current round, it can create overhead on the validator such that a backlog accumulates over time, until the validators start dropping packets: Each round is executed quickly, and one validator may still be processing attack messages when it becomes again the leader in the next iteration. Then, the more validators, the more time it takes to build up the overhead backlog across the whole committee.

Interestingly, there does not seem to be a difference between internal and external adversaries, as shown in Fig. 9 in the Appendix. This is likely because the main factor of attack success is not the immediate compromise of each successive leader—for which the timely dissemination of information about the current

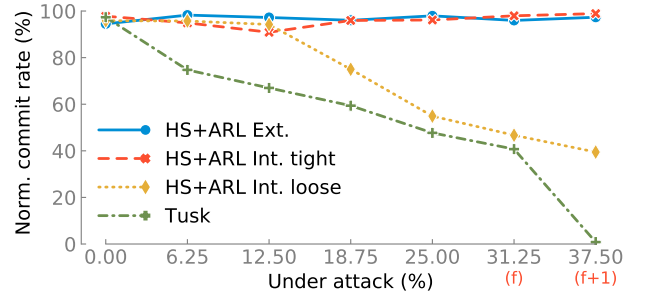


Figure 6: Fixed-subset attack with defenses. Replicating the scenario of Fig. 4 with defenses deployed. Tusk $\dashv\vdash$ resists attacks better than HotStuff (HS). Authentication and rate-limiting (ARL) almost completely removes threats, except when the rate-limiting threshold is poorly chosen $\dashv\vdash$ (see Fig. 11). “Int.” and “Ext.” denote internal and external adversaries.

leader is crucial—but rather the slower overwhelming of each of the validators until the leaders start dropping packets.

6 Attacking Asynchronous Consensus Protocols

The experiments in the previous section highlight the susceptibility of leader-based BFT partially-synchronous consensus protocols such as HotStuff to liveness attacks. We now investigate whether other protocol designs are more DoS resilient by attacking asynchronous consensus protocols.

As introduced in §2.2, Tusk is one such asynchronous consensus protocol that completely forgoes the use of a leader to drive consensus. Previous work has shown that Tusk does not incur a significant performance penalty compared to HotStuff, and can process thousands of transactions per second with low delay. Therefore, Tusk may be a practical and DoS-resilient alternative to leader-based consensus protocols. We thus run signature flood attacks against Tusk, similar to the ones we used to disrupt HotStuff.

6.1 Tusk Implementation

We use the reference Tusk implementation in Rust¹⁰ (also containing Narwhal, §2.2), which is comparable in design and performance to the HotStuff implementation. Similarly to the HotStuff implementation, Tusk uses TCP to achieve reliable point-to-point channels, necessary to correctly implement the distributed system abstractions. The VPSes, network, and adversary setup are identical to the attacks against HotStuff (§4.2). To compare Tusk and HotStuff with similar low-overhead workloads, we set the transaction rate to 10 000 transactions per second.

This Tusk research implementation avoids the complexity of using a shared random coin to select the certificate to be committed, and instead proceeds in round-robin fashion. We do not exploit this detail for attacks, as production-ready implementations of Tusk would not introduce this attack vector.

¹⁰<https://github.com/asonnino/narwhal>

6.2 Fixed-subset Attacks on Tusk

The Certificate Signature Flood. Tusk relies on the exchange of Certificate messages across validators, which act both as votes for previous proposals, as well as proposals for new blocks to be committed. For each certificate, validators need to verify a signature to authenticate the message. Similarly to the Block attacks on HotStuff, an internal adversary may further abuse the many signatures in the Certificate to inflict an even higher overhead.

Results. We attack a fixed subset of Tusk validators of size t , with t in $\{1, 2, 3, 4, 5, 6\}$, in a committee of 16 ($f = 5$). The adversary also controls 16 machines. For each value of t we run the experiment twice and compute an average of the results, leading to 12 experiments in total.

Figure 6 shows that Tusk's normalized commit rate under attack $\dashv\vdash$ degrades more gently with the increasing number of targets t compared to HotStuff. Only attacks on $f + 1$ validators achieve a full loss of liveness. Even for attacks on $t = f$ validators, the commit rate is still above 40% of the rate in normal operation. This is as expected: Only if the validator that is selected to commit was under attack the consensus loses a round of commits, while the remaining rounds are unaffected. Moreover, the other Tusk validators continue to advance the DAG by propagating user transactions asynchronously. The next successful leader will then commit its entire sub-DAG, including the transactions that the leader under attack did not manage to commit. As a result, the degradation in throughput is only due to the reduced capacity of the Tusk validators under attack. Further, since Tusk is an asynchronous protocol, it advances at network speed and does not suffer from the long timeouts that lower HotStuff's commit rate.

7 Network-Layer Defenses

Our comparison of HotStuff and Tusk under a DoS attacks shows the advantage of running an asynchronous, leader-less consensus protocol in terms of throughput and liveness. However—even though Tusk retains liveness for attacks on less than $f + 1$ validators—the adversary is still able to significantly impair its throughput. Moreover, the adversary can target more than f validators with little additional effort, completely halting progress.

The core issue is that consensus protocols rely on the assumption of *point-to-point authenticated and always-available links* for their liveness guarantees. Since this network abstraction is hard to implement in practice, the adversary can disrupt communication channels between the validators and no liveness guarantee can be achieved. It is therefore clear that a comprehensive solution to DoS attacks on consensus protocols must also consider network-level defenses that can better implement this abstraction.

Thus, we explore the network security literature for effective DoS mitigations. We start by considering traditional DoS defenses, and explain why they may be insufficient for the protection of distributed consensus protocols. Then, we draw from recent works to propose new defenses specifically tailored for consensus protocols. We conclude with a set of experiments testing the effectiveness of these mitigations.

7.1 Defense System Requirements

Blockchains feature unique communication requirements:

- *High availability:* This includes protection against naturally-occurring failures and against routing and DoS attacks.
- *Decentralized operation:* Validators must be fully distributed, without relying on any single entity. This must include validator hosting hardware and networks. Further, to avoid vendor lock-in, all systems should be open source.
- *Decentralized economics:* Validator deployment must be affordable, to guarantee the widest possible deployment and incentivize decentralization.

Because of these peculiar requirements, most of the existing DoS defenses are ill-suited to protect distributed consensus.

7.2 Traditional Defenses Are Insufficient

We broadly survey the most common classes of DoS defenses, and discuss why they violate the requirements above.

IP-based Filtering. Within our adversary model, IP-based traffic filtering is ineffective. If an IP-blocklist is deployed, the adversary can quickly rotate bots and send traffic from different IP addresses. If, on the other hand, an allowlist is implemented, the adversary can bypass it by spoofing source addresses, possibly framing a honest validator for an attack.

Cloud-Based DoS Protection. Several commercial offerings [3, 23, 57], as well as research papers [31, 38, 50], offload the filtering of adversarial traffic to the cloud. However, cloud-based protections are not applicable to the defense of consensus protocols. First, these services are expensive, and are only offered by a handful of providers, favoring centralization. Second, the cloud provider needs to have specific filtering rules that separate adversarial traffic from legitimate consensus traffic. With attack packets coming from possibly thousands of hosts in a botnet, and without an established way to source-authenticate traffic, the adversary's traffic is hard to distinguish from honest traffic. Finally, since our attacks are relatively low-rate compared to standard volumetric DoS, and can be spread across multiple sources, cloud filters cannot rely on the traffic volume to detect whether an attack is underway.

Overprovisioning. Another usual avenue for DoS mitigation is overprovisioning, where entities—validators in this case—provision their compute, storage, and network to withstand the highest loads. If the protocol can horizontally scale and make use of these additional resources, attacks become increasingly difficult. This solution, however, requires massive resources, and thus favors centralization. Blockchains are designed to financially incentivize validators, and thus need to run on relatively cheap hardware.

TLS & VPNs. The committee could leverage the pre-shared consensus keys to set up authenticated tunnels between validators. These tunnels could be implemented, e.g., by long-lasting TLS connections or virtual private network (VPN) connections. However, these protocols rely on complex connection handshakes and keep per-connection state, which have been repeatedly exploited for DoS attacks in the past. Recent work has shown that many major VPN solutions (including WireGuard and OpenVPN) suffer from debilitating, low-rate DoS attacks that prevent new connections, and even force established connections to be dropped [63]. Further, the TCP connections over which TLS is customarily transported have been recently shown to be susceptible to cross-layer hijacking and connection reset attacks [32, 33].

Even if perfectly patched, and using pre-shared keys to avoid DoS attacks on the handshake, these protocols alone are only sufficient to discard attack traffic from external adversaries. Such a defense would certainly be an improvement over current practices, but internal adversaries could still bypass it by using legitimate keys.

Building on these observations, we propose a defense that combines (i) lightweight symmetric-key authentication, to protect against signature floods, together with (ii) per-key rate-limiting, to stop internal adversaries from generating too much traffic. This system is moreover implemented at the *network layer*, so that state exhaustion attacks against the transport layer are also not possible.

7.3 Consensus-specific Defenses: Source Authentication and Rate Limiting

We now outline a more principled, consensus-specific defense against DoS attacks, that matches the requirements in §7.1.

During flooding-based DoS attacks, including the ones presented in this work, an adversary overwhelms a service by sending a high number of requests. By monitoring and limiting the number of requests received by the validator, we can ensure that any received request can be processed during a reasonable time frame. However, rate-limiting is only effective if the limit is enforced on authenticated traffic from validators—otherwise, an adversary may spoof the addresses of legitimate validators, bypass the rate-limiting, or even exhaust the rate limit for other legitimate clients causing their traffic to be dropped. Therefore, deploying a source-authentication and rate-limiting (ARL) system in front of each validator can prevent the attacks presented in this paper, while respecting the decentralization demands of consensus protocols. Given an ARL system, validators can (i) drop packets from external adversaries before they reach the consensus logic, and (ii) impose strict rate limits on the consensus traffic coming from other validators, thus preventing attacks from internal adversaries. Further, note that an ARL-based defense system is *decentralized*, as validators can run and manage their instances independently.

To test the effectiveness of this measure, we deploy an ARL system in front of each validator. Crucially, this subsystem should not open new avenues for DoS attacks. Therefore, our prototype implementation is based on symmetric-key cryptography, and can thus authenticate packets and enforce rate limits with minimal computational resources—effectively at line rate. The functionality of this system is minimal by design, to further enhance efficiency and reduce the attack surface.

We now provide an overview of the ARL’s operation, and then test its effectiveness as DoS mitigation for validators.

The ARL Pipeline. The ARL is composed of two distinct components: a packet authentication system and a rate limiter. The packet authentication system unambiguously attributes each incoming packet to a source host or to a “best-effort” unauthenticated category. Packets that pass authentication are then forwarded to the rate limiter, which ensures that sources do not exceed their pre-defined rates. The best-effort category is also throttled. Our ARL prototype is not sophisticated enough to protect against replay attacks—whereby an adversary captures and sends replicas of legitimate packets in large volumes. However, *duplicate suppression*

systems have been studied in the literature and could be directly applied in ARL as a third stage in the pipeline [48].

Symmetric Key Establishment. To use the ARL, each pair of consensus validators A and B must share a pair of symmetric keys $K_{A \rightarrow B}$ and $K_{B \rightarrow A}$, where the direction of the arrow denotes the direction of communication.

Discussing the key exchange in depth is beyond the scope of this paper. However, we note that validators must already share public keys with each other to participate in the consensus protocol (most importantly, to sign Votes, Blocks, and Certificates). Commonly, the public keys of validators are shared by encoding them in transaction blocks when a new committee is elected. The root of trust is then the *genesis block*—the first block of transactions in the state machine—which contains the keys for the initial committee. Validators can therefore leverage these pre-shared consensus keys to authenticate a standard key negotiation protocol—e.g., Diffie-Hellman [26]—and derive symmetric keys for the ARL.

After the first exchange—which occurs, e.g., when a new validator joins the protocol—all subsequent re-keying can be protected by ARL. The key exchanges can be performed before the expiry of the previous key, so that the key requests can be rate-limited under ARL: When the previous keys expire, the validators are already in possession of a fresh pair and can continue using the ARL undisturbed. This simple re-keying strategy prevents flooding attacks on the key exchange, drastically limiting the attack surface for DoS.

Finally, we need to consider the overhead of storing and retrieving the symmetric keys at line rate. Since in every committee the number of validators does not normally exceed the hundreds, keeping the keys in cache is feasible and allows a fast and efficient retrieval. However, in case the number of validators is too high to store all keys in cache, a system such as PISKES [58] can be deployed to decrease the overhead of fetching the keys from memory at the destination.

Packet Authentication Details. For every packet sent from validator A to B , ARL adds an additional header

$$\text{HDR} = (A, \text{TS}, l_{\text{pkt}}),$$

where TS is a timestamp and l_{pkt} is the packet length. Then, ARL computes a cryptographic authentication tag over the HDR and a hash of the payload:

$$\text{PH} = H(\text{payload}), \quad \text{TAG} = \text{MAC}_{K_{A \rightarrow B}}(\text{HDR}, \text{PH}).$$

Both TAG and PH are added to the packet, which is then forwarded towards the destination. The destination B then uses HDR, PH, and the pre-shared $K_{A \rightarrow B}$ to recompute the MAC, and matches against the received TAG. If they are the same, the destination then recomputes $H(\text{payload})$ and compares it to PH to verify the integrity of the payload. Informally, if the procedure succeeds, then validator A must have sent the packet, as only an entity with the right $K_{A \rightarrow B}$ could have produced the correct MAC. Thus, ARL achieves per-packet source authentication.

Notice that sending both PH and l_{pkt} to the destination also improves the attack resilience of the protocol. Without this information, the adversary could replace the existing payload with an MTU-sized payload, which then the destination would have to compute a hash of. Instead, the destination first checks that the length of the received packet is equal to l_{pkt} , mitigating this attack vector.

8 Attacking ARL-Protected Consensus

We experiment with attacks against HotStuff protected by the ARL, and compare the resilience of this consensus-specific defense with the unmodified HotStuff and Tusk protocols.

ARL Attack Scenarios. We consider three attack scenarios, obtained by varying (i) the adversary’s knowledge of the ARL (aware/unaware of the defense); and (ii) the adversary’s relation to the committee (internal/external):

- *ARL-unaware external adversary:* This adversary just targets a fixed subset of validators with a Vote signature flood, without considering the presence of ARL. Since there is no ARL header in the attack packets, they will be discarded without performing the authentication step.
- *ARL-aware external adversary:* This attack is similar to the previous, with the distinction that an (unauthentic) ARL header is added to packets, forcing the ARL application on the validators to perform (and fail) packet authentication.
- *ARL-aware internal adversary:* In this final and most powerful attack, the adversary is internal to the committee and therefore holds valid pre-shared symmetric keys with the validators. It then performs a Vote signature flood (for comparison with the attacks above), this time successfully authenticating the attack packets. The rate limiter, however, caps the amount of attack traffic that reaches the consensus logic at the validators.

ARL Deployment. ARL is implemented in C using the DPDK high-speed packet processing framework¹¹, following the high-level system design in §7.3, and based on an open-source project.¹² ARL is deployed on all honest and compromised validators, and performs filtering and rate-limiting for all incoming packets, while adding the ARL header to all outgoing packets. All packets that are not authenticated or exceed the rate limit are immediately discarded. Each validator rate-limits traffic from other validators to a fixed maximum rate threshold in Mbps. We assume that an internal adversary may have access to one of the ARL authentication keys. Therefore, the *aggregate* traffic from all the adversary machines is rate-limited to the same fixed threshold. Note that if the adversary controls more Byzantine validators—up to f —it will trivially be able to send at as much as f times the rate of the threshold. We therefore experiment with the least powerful adversary (one key); when provisioning to withstand an adversary with f keys, the rate-limiting threshold must be chosen to be $1/f$ the threshold of the single-key adversary to similarly limit the adversary’s attack power.

The Rate-Limiting Threshold. For the rate limits, ARL differentiates between consensus protocol and mempool protocol packets. This is because the mempool has much higher bandwidth requirements than the consensus. If consensus and mempool were to be rate-limited together, the threshold would have to be set to tens or hundreds of Mbps, and an internal adversary may then use this wide bandwidth allowance to target the consensus. Choosing appropriate rate-limits—in this case for the consensus port—is critical, especially when considering internal adversaries: Too low, and the honest validators cannot communicate; too high, and the adversary may still be able to send enough traffic to overwhelm the validators.

To determine an appropriate value for the rate-limiting threshold, we run an experiment with 8 validators and 1 internal adversary controlling 8 attacking machines, and change the rate-limit threshold in exponential increments. We look at the resulting normalized commit rate to gauge the effectiveness of the attack. We see in Fig. 11 in the Appendix that the adversary must be able to send at least 8 Mbps to achieve an impact, while with 16 Mbps the adversary is able to almost completely stall the consensus. We, therefore, test the effects of choosing the wrong rate limit threshold, and run two experiments with a “tight” threshold of 2 Mbps, and a “loose” threshold of 10 Mbps that allows for enough attack traffic to reach the consensus logic and create a moderate amount of disruption.

Results. As before, the consensus committee consists of 16 validators, and the adversary controls 16 machines. Our results indicate that ARL is very effective in preventing attacks against HotStuff. The attack traffic from external adversaries is immediately discarded, as either packets do not include the ARL header (ARL-unaware external adversary), or the authentication MAC is invalid (ARL-aware external adversary). In Fig. 6 we only show the latter \rightarrow , as in both cases the normalized commit rate is $\approx 100\%$.

When considering internal adversaries—which possess valid ARL keys and are therefore necessarily ARL-aware—we see that indeed the choice of rate-limiting threshold makes a difference. With a “tight” threshold of 2 Mbps \rightarrow , the adversary cannot influence the consensus throughput. On the other hand, with the “loose” threshold of 10 Mbps \rightarrow , the adversary can increase load on the validators and lower the throughput. We can see two runs of the consensus with different thresholds in Fig. 10 in the Appendix: after some time, the constant stream of signatures accumulates and starts forcing some dropped packets, delaying the consensus.

9 Discussion

Throughout the paper, we have verified the following hypotheses on the resilience of consensus protocols to DoS attacks:

- In HotStuff, a leader-based consensus protocol, the round leader represents a single point of failure, and this weakness can be exploited by an adversary in practice. Most notably, the adversary can disrupt consensus independently of the size of the committee, violating the core availability tenet that the larger the committee, the costlier an attack should be. This problem is exacerbated by the reliance on timers: If the leader crashes, no progress can be made until the timers expire and the leader is rotated.
- Tusk, an asynchronous consensus protocol, is more resistant to DoS attacks because it does not rely on timers to advance consensus. The consensus throughput may still decrease under attack, but liveness is maintained as long as a quorum of honest validators is active.
- The introduction of network-layer defenses, such as the authentication and rate-limiting system (ARL), greatly increases the survivability of HotStuff. Figure 6 clearly highlights that ARL can improve the resilience of HotStuff validators to the point that their throughput under attack is better than Tusk’s, despite the weakness introduced by the single leader and the use of timers.

Future work is needed to evaluate the combination of asynchronous consensus protocols with ARL, although preliminary results indicate that ARL is a suitable protection in this case as well.

¹¹<https://www.dpdk.org/>

¹²<https://github.com/netsec-ethz/lightning-filter>

Attack implementation relevance. In the experiments, we use the reference Hotstuff and Tusk implementations developed at Facebook/Meta for research purposes. It uses the same core consensus logic as the latest production DiemBFT codebase (DiemBFTv4), running the same persistent storage, crypto, and network stack. Most importantly, there are no additional DoS protections in the production DiemBFT codebase. We therefore believe that the results presented in the paper are representative of the most advanced consensus engine implementations. Moreover, well-engineered prototypes are much simpler in terms of functionality than a fully-fledged blockchain, and therefore provide an improvement on the attack resilience of a blockchain: The additional overhead of RPC endpoints, transaction execution engines, etc., only increases the DoS attack surface. Our attacks are then targeted against *consensus engines*, rather than blockchains, and therefore provide an upper bound of the resilience of BFT systems.

Generalizing the attacks. We focused our DoS attacks against HotStuff and Tusk. This section discusses how those attacks generalize to other consensus protocols.

Synchronous protocols such as Sync-HotStuff [1] are subject to the same attacks described in this paper. Unlike their partially-synchronous counterparts, synchronous protocols can tolerate up to half faulty validators. Thus, attacking a fixed subset of validators (§5.2) requires more adversarial resources; leader-tracking attacks (§5.3) remain identical. DoS attacks however are a greater risk for synchronous protocols: while DoS attacks against partially-synchronous protocols only halt consensus (liveness loss), the same attacks against synchronous protocols may result in honest validators ending up with different commit sequences (safety violation).

Our results on Tusk will generalize to other asynchronous consensus protocols [2, 12, 24, 42, 52]. In general, asynchronous protocols do not directly depend on leaders to drive the protocol nor on timeouts to make progress. As a result, these protocols do not suffer from leader-tracking attacks (§5.3). Further, DoS attacks against a fixed subset of validators are less effective (as demonstrated in §8), as they are designed to operate under a stronger adversary model.

Finally, we note that our results partly apply to partially-synchronous protocols that do not directly rely on a leader to drive consensus. E.g., in Algorand [37], the leader is still detectable and can be targeted, as it broadcasts block proposals. However, the leader selection is unpredictable and the communication is based on gossip messages, drastically lowering the overall dependence on the leader. In Bullshark [39] (i) the leader cannot be distinguished through traffic patterns, and (ii) the election sequence can be randomized. Nonetheless, both these protocols still rely on timers for their operation, which may still cause sharp drop in throughput and loss of liveness under attack. We leave the evaluation of the DoS resilience of these defense mechanisms to future work.

10 Related Work

In addition to the related work on BFT consensus protocols and DoS attacks (§2), and traditional DoS defenses (§7.2), presented in previous sections, we add here other related publications to contextualize our contribution.

Attacks on Blockchains. To the best of our knowledge, Spiegelman and Rinberg [62] are the first and only to analyze the effects

of DoS attacks on HotStuff. While their work focuses on a protocol that turns partially-synchronous protocols into fully asynchronous, their evaluation shows that a flood of client requests targeted at the leader may halt consensus in a small HotStuff deployment.

Outside DoS and quorum-based consensus, the blockchain space presents many attack vectors that an adversary may use to attack the safety or liveness of the protocols [9, 30, 36]. Proof-of-work blockchains, such as Bitcoin [55] and Ethereum [66], are vulnerable to protocol-level attacks that lower their security threshold below 51%. The most notable are selfish mining [59], stubborn mining [56], Fork After Withholding (FAW) attack [45], and eclipse attack [41]. Further, these blockchains have been found to be vulnerable to BGP hijacking attacks [6]. Proof-of-stake blockchains [13, 43] open up new attacks compared to proof-of-work consensus [9]: the nothing-at-stake attack; the grinding attack; and the long-range attack [8].

Other DoS Defenses. Zargar et al. [68] published a systematization of existing DoS defenses; given the wealth of work on the topic, we review relevant recent proposals. A first example are routing-based defenses [60], whereby autonomous systems under attack redirect traffic through BGP announcements. However, the effectiveness of these systems has been debated [65]. In the context of Bitcoin, the SABRE system was proposed to mitigate BGP hijacking attacks [5]; path-stable Internet architectures such as SCION [19] are a solution to these attacks, as off-path entities cannot influence the routing process. Finally, capability-based defenses, which grant access to network resources based on cryptographic tokens embedded inside packets [4, 40], could also be used to defend consensus nodes, and even protect against more powerful attacks such as Coremelt [64], where the adversary targets the network infrastructure instead of the endpoint. These systems bear similarities to our ARL implementation at the endpoints. However, they require in-network support by forwarding elements, increasing their deployment complexity. Therefore, studying the effectiveness of capability-based defenses in protecting consensus protocols is left as future work.

11 Conclusion

As blockchains are maturing to be integrated into critical infrastructure, ensuring their high availability is paramount. In this paper, we experimentally show the effectiveness of DoS attacks against current state-of-the-art consensus protocols, and evaluate practical defenses. In the case of leader-based and partially synchronous consensus protocols—exemplified in our experiments by HotStuff—the cost of an effective attack on liveness is independent of the committee size. The leader is a vulnerable single point of failure, which can be exploited by an adversary to swiftly disrupt the consensus. We then test whether modern fully-asynchronous consensus protocols, represented by Tusk, are more resilient to the attacks. Indeed, we empirically show the increased DoS resistance of these protocols—which do not require an active leader to proceed—although their throughput is still impacted by the attacks. Finally, we analyze the effects of deploying lower-layer DoS defenses at validators. We demonstrate that a combination of source authentication and rate-limiting almost entirely mitigates our attacks, while respecting the decentralization requirements of consensus.

The breadth and depth of the attack analysis presented in this paper is, to the best of our knowledge, novel in the study of BFT consensus protocols. Beyond the immediate relevance of our results, we hope this work will highlight the challenges of deploying consensus protocols in real networks, and that the experimental methodology defined herein will help guide future evaluations.

Ethical Considerations. We have carefully followed a responsible disclosure process.

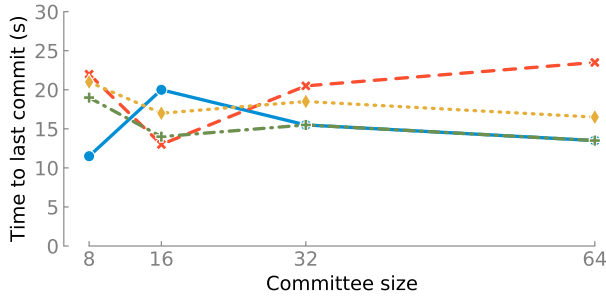
Acknowledgements

This work was mostly realized while Alberto Sonnino and Lefteris Kokoris-Kogias were employed at Meta. We gratefully acknowledge support for this project from ETH Zurich and Mysten Labs.

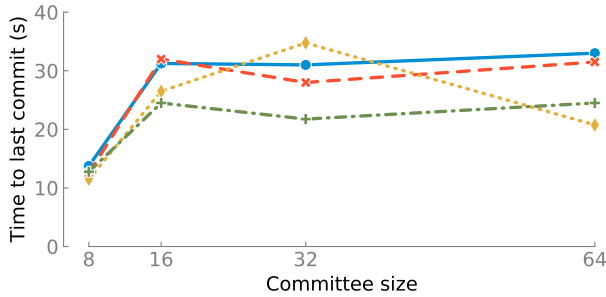
References

- [1] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. 2020. Sync hotstuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 106–118.
- [2] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. 2019. Asymptotically optimal validated asynchronous byzantine agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. 337–346.
- [3] Amazon. 2022. Amazon AWS Shield. <https://aws.amazon.com/shield/>.
- [4] Tom Anderson, Timothy Roscoe, and David Wetherall. 2004. Preventing Internet denial-of-service with capabilities. *ACM SIGCOMM Computer Communication Review (CCR)* 34, 1 (2004). <https://doi.org/10.1145/972374.972382>
- [5] Maria Apostolaki, Gian Marti, Jan Müller, and Laurent Vanbever. 2019. SABRE: Protecting Bitcoin against Routing Attacks. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society. <https://www.ndss-symposium.org/ndss-paper/sabre-protecting-bitcoin-against-routing-attacks/>
- [6] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. 2017. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *2017 IEEE symposium on security and privacy (SP)*. IEEE, 375–392.
- [7] Aptos. 2022. Building the safest and most scalable Layer 1 blockchain. <https://aptoslabs.com>.
- [8] Sarah Azouvi, George Danezis, and Valeria Nikolaenko. 2020. Winkle: foiling long-range attacks in proof-of-stake systems. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. 189–201.
- [9] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. 2017. Consensus in the age of blockchains. *arXiv preprint arXiv:1711.03936* (2017).
- [10] Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, and Alberto Sonnino. 2019. State machine replication in the Libra blockchain. *The Libra Assn., Tech. Rep* (2019).
- [11] Stephanie Bayer and Jens Groth. 2012. Efficient zero-knowledge argument for correctness of a shuffle. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 263–280.
- [12] Michael Ben-Or. 1983. Another advantage of free choice (extended abstract) completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*. 27–30.
- [13] Iddo Bentov, Rafael Pass, and Elaine Shi. 2016. Snow White: Provably Secure Proofs of Stake. *IACR Cryptol. ePrint Arch.* 2016, 919 (2016).
- [14] The Celo Blog. 2022. Celo Sets Sights On Becoming Fastest EVM Chain Through Collaboration With Mysten Labs. <https://medium.com/celoorg/celo-sets-sights-on-becoming-fastest-evm-chain-through-collaboration-with-mysten-labs-e88b426aee83>.
- [15] Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. 2020. Single secret leader election. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. 12–24.
- [16] Gabriel Bracha. 1987. Asynchronous Byzantine agreement protocols. *Information and Computation* 75, 2 (1987), 130–143.
- [17] Christian Cachin, Rachid Guerraoui, and Luis Rodrigues. 2011. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media.
- [18] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Trans. Comput. Syst.* 20, 4 (nov 2002), 398–461. <https://doi.org/10.1145/571637.571640>
- [19] Laurent Chauat, Markus Legner, David Basin, David Hausheer, Samuel Hitz, Peter Müller, and Adrian Perrig. 2022. *The Complete Guide to SCION. From Design Principles to Formal Verification*. Springer International Publishing AG. <https://link.springer.com/book/10.1007/978-3-031-05288-0>
- [20] Catalin Cimpanu. 2021. Belgium's government network goes down after massive DDoS attack. <https://archive.ph/vZ5Oe>. [Archived: 2023-04-25].
- [21] Cision. 2022. Sommelier Partners with Mysten Labs to make the Cosmos blockchain the fastest on the planet. <https://www.prnewswire.com/news-releases/sommelier-partners-with-mysten-labs-to-make-the-cosmos-blockchain-the-fastest-on-the-planet-301381122.html>.
- [22] CloudFlare. 2022. Comprehensive DDoS protection. <https://www.cloudflare.com/ddos/>.
- [23] Cloudflare. 2022. Comprehensive DDoS protections. <https://www.cloudflare.com/en-gb/ddos/>.
- [24] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*. 34–50.
- [25] Diem. 2021. Welcome to the Diem project. <http://diem.com>.
- [26] W. Diffie and M. Hellman. 1976. New directions in cryptography. *IEEE Transactions on Information Theory* 22, 6 (1976), 644–654. <https://doi.org/10.1109/TIT.1976.1055638>
- [27] Danny Dolev, Michael J. Fischer, Rob Fowler, Nancy A. Lynch, and H. Raymond Strong. 1982. An efficient algorithm for byzantine agreement without authentication. *Information and Control* 52, 3 (1982), 257–274. [https://doi.org/10.1016/S0019-9958\(82\)90776-8](https://doi.org/10.1016/S0019-9958(82)90776-8)
- [28] Justin Drake. 2019. Low-overhead secret single-leader election. <https://ethresear.ch/t/low-overhead-secret-single-leader-election/5994>.
- [29] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)* 35, 2 (1988), 288–323.
- [30] Ittay Eyal and Emin Gün Sirer. 2018. Majority is Not Enough: Bitcoin Mining is Vulnerable. *Commun. ACM* 61, 7 (jun 2018), 95–102. <https://doi.org/10.1145/3212998>
- [31] S.K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey. 2015. Bohatei: Flexible and elastic DDoS defense. *USENIX Security Symposium* (2015), 817–832.
- [32] Xuewei Feng, Chuanpu Fu, Qi Li, Kun Sun, and Ke Xu. 2020. Off-Path TCP Exploits of the Mixed IPID Assignment. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (Virtual Event, USA) (CCS '20)*. Association for Computing Machinery, New York, NY, USA, 1323–1335. <https://doi.org/10.1145/3372297.3417884>
- [33] Xuewei Feng, Qi Li, Kun Sun, Chuanpu Fu, and Ke Xu. 2022. Off-Path TCP Hijacking Attacks via the Side Channel of Downgraded IPID. *IEEE/ACM Transactions on Networking* 30, 1 (2022), 409–422. <https://doi.org/10.1109/TNET.2021.3115517>
- [34] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM* 32, 2 (1985), 374–382.
- [35] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. 2021. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. *arXiv preprint arXiv:2106.10362* (2021).
- [36] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srđjan Capkun. 2016. On the Security and Performance of Proof of Work Blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (Vienna, Austria) (CCS '16)*. Association for Computing Machinery, New York, NY, USA, 3–16. <https://doi.org/10.1145/2976749.2978341>
- [37] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*. 51–68.
- [38] Y. Gilad, A. Herzberg, M. Sudkovich, and M. Góberman. 2016. CDN-ondemand: An affordable DDoS defense via untrusted clouds. *Network and Distributed System Security Symposium - NDSS* (2016), 1–15.
- [39] Neil Girdharan, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Bullshark: Dag bft protocols made practical. *arXiv preprint arXiv:2201.05677* (2022).
- [40] Giacomo Giuliani, Dominik Roos, Marc Wyss, Juan Angel García-Pardo, Markus Legner, and Adrian Perrig. 2021. Colibri: A Cooperative Lightweight Inter-domain Bandwidth-Reservation Infrastructure. In *International Conference on emerging Networking EXperiments and Technologies (CoNEXT '21)*. <https://doi.org/10.1145/3485983.3494871>
- [41] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, Washington, D.C., 129–144. <https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman>
- [42] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. 2021. All You Need is DAG. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC'21)*. Association for Computing Machinery. <https://doi.org/10.1145/3465084.3467905>
- [43] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynkov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual international cryptology conference*. Springer, 357–388.
- [44] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing Bitcoin Security and Performance

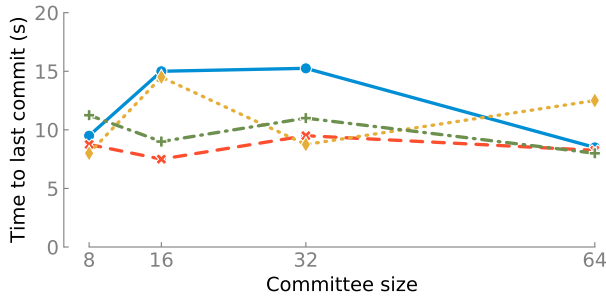
- with Strong Consistency via Collective Signing. *CoRR* abs/1602.06997 (2016). arXiv:1602.06997 <http://arxiv.org/abs/1602.06997>
- [45] Ujin Kwon, Dohyun Kim, Yunmok Son, Eugene Y. Vasserman, and Yongdae Kim. 2017. Be Selfish and Avoid Dilemmas: Fork After Withholding (FAW) Attacks on Bitcoin. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 195–209. <https://doi.org/10.1145/3133956.3134019>
- [46] Mysten Labs. 2022. Build without Boundaries. <https://sui.io>.
- [47] Mysten Labs. 2022. The Sui Smart Contract Platform. <https://github.com/MystenLabs/sui/blob/main/doc/paper/sui.pdf>.
- [48] Taeho Lee, Christos Pappas, Adrian Perrig, Virgil Gligor, and Yih-Chun Hu. 2017. The Case for In-Network Replay Suppression. In *ACM Asia Conference on Computer and Communications Security (ASIACCS)*. <https://doi.org/10.1145/3052973.3052988>
- [49] Benoît Libert, Marc Joye, and Moti Yung. 2016. Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. *Theoretical Computer Science* 645 (2016), 1–24.
- [50] Z. Liu, H. Jiny, Y.-C. Hu, and M. Bailey. 2016. Middlepolice: Toward enforcing destination-defined policies in the middle of the internet. *ACM Conference on Computer and Communications Security 24-28-October-2016* (2016), 1268–1279. <https://doi.org/10.1145/2976749.2978306>
- [51] Mary Maller. 2020. Provable Single Secret Leader Election. <https://ethresear.ch/t/provable-single-secret-leader-election/7971>.
- [52] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 31–42.
- [53] F Mölder, KP Jablonski, B Letcher, MB Hall, CH Tomkins-Tinch, V Sochat, J Forster, S Lee, SO Twardziok, A Kanitz, A Wilm, M Holtgrewe, S Rahmann, S Nahnsen, and J Köster. 2021. Sustainable data analysis with Snakemake [version 1; peer review: 1 approved, 1 approved with reservations]. *F1000Research* 10, 33 (2021). <https://doi.org/10.12688/f1000research.29032.1>
- [54] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review* (2008), 21260.
- [55] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review* (2008), 21260.
- [56] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. 2016. Stubborn Mining: Generalizing Selfish Mining and Combining with an Eclipse Attack. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. 305–320. <https://doi.org/10.1109/EuroSP.2016.32>
- [57] Radware. 2022. Radware's DefensePro DDoS Protection. <https://www.radware.com/products/defensepro/>.
- [58] Benjamin Rothenberger, Dominik Roos, Markus Legner, and Adrian Perrig. 2020. PISKES: Pragmatic Internet-Scale Key-Establishment System. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (Taipei, Taiwan) (ASIA CCS '20)*, Association for Computing Machinery, New York, NY, USA, 73–86. <https://doi.org/10.1145/3320269.3384743>
- [59] Ayelet Sapirshstein, Yonatan Sompolsky, and Aviv Zohar. 2015. Optimal Selfish Mining Strategies in Bitcoin. In *Financial Cryptography*.
- [60] Jared M Smith and Max Schuchard. 2018. Routing Around Congestion: Defeating DDoS Attacks and Adverse Network Conditions via Reactive BGP Routing. In *IEEE Symposium on Security and Privacy (S&P)*. <https://doi.org/10.1109/sp.2018.00032>
- [61] Alberto Sonnino. 2022. 2-Chain Variant of the HotStuff Consensus Protocol. <https://github.com/asonnino/hotstuff>.
- [62] Alexander Spiegelman, Arik Rinberg, and Dahlia Malkhi. 2021. ACE: Abstract Consensus Encapsulation for Liveness Boosting of State Machine Replication. In *24th International Conference on Principles of Distributed Systems (OPODIS 2020)*. <https://doi.org/10.4230/LIPIcs.OPODIS.2020.9>
- [63] Fabio Streun, Joel Wanner, and Adrian Perrig. 2022. Evaluating Susceptibility of VPN Implementations to DoS Attacks Using Adversarial Testing. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*. <https://doi.org/10.14722/ndss.2022.24043>
- [64] Ahren Studer and Adrian Perrig. 2009. The coremelt attack. In *European Symposium on Research in Computer Security*. Springer, 37–52.
- [65] Muoi Tran, Min Suk Kang, Hsu-Chun Hsiao, Wei-Hsuan Chiang, Shu-Po Tung, and Yu-Su Wang. 2019. On the Feasibility of Rerouting-based DDoS Defenses. In *IEEE Symposium on Security and Privacy (S&P)*. <https://doi.org/10.1109/SP.2019.00055>
- [66] Gavin Wood. 2022. Ethereum: A Secure Decentralised Generalised Transaction Ledger. <https://ethereum.github.io/yellowpaper/paper.pdf>.
- [67] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. 347–356.
- [68] S. T. Zargar, J. Joshi, and D. Tipper. 2013. A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks. *IEEE Communications Surveys & Tutorials* 15, 4 (2013). <https://doi.org/10.1109/SURV.2013>



(a) Static leader attack (external adversary).



(b) Leader tracking attack (external adversary).



(c) Leader tracking attack (internal adversary).

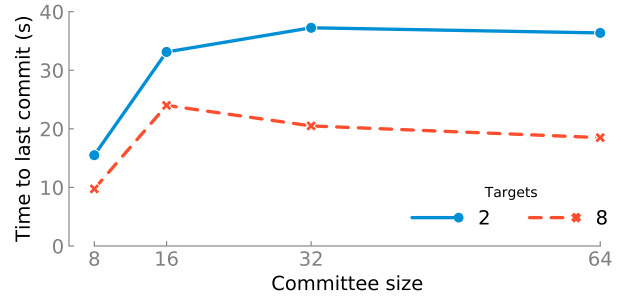
Attackers
 ● 8 * 16 ◆ 32 + 64

Figure 7: Breakdown of the leader-tracking attacks by number of adversaries.

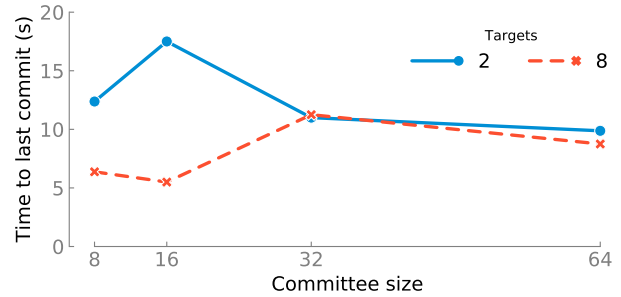
A Additional Results

We provide here additional results to further help in understanding the effects of the attacks presented in the paper.

HotStuff Leader-Tracking Attacks. Figure 7 shows the breakdown by adversary size for the static, external, and internal leader-tracking attacks (◆, *, and ● respectively in Fig. 2); similarly, Fig. 8 shows the breakdown by number of targets.



(a) Leader tracking attack (external adversary).



(b) Leader tracking attack (internal adversary).

Figure 8: Breakdown of the leader-tracking attacks by number of targets in the committee.

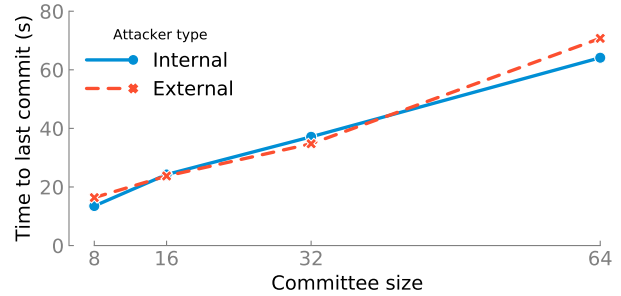
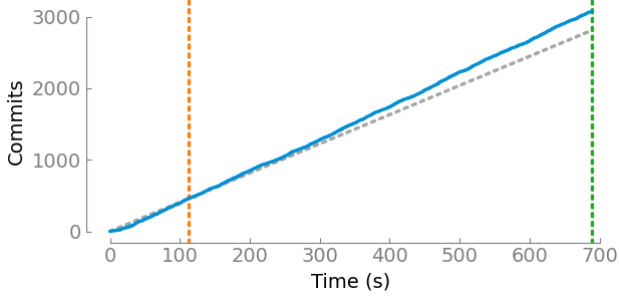
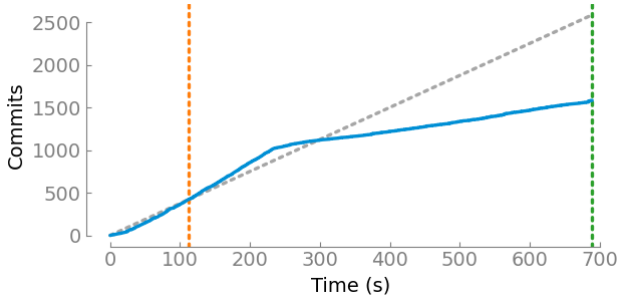


Figure 9: Unpredictable leader-tracking attacks.

Types of Unpredictability. As described in §5.4, unpredictability in the leader-election procedure can be a mitigation to leader-tracking attacks. Figure 9 breaks down the results by *type* of unpredictability.



(a) “Tight” threshold (2 Mbps).



(b) “Loose” threshold (10 Mbps)

Figure 10: Effects of the ARL rate-limiting threshold on Hot-Stuff. With the “loose” rate limit, the internal adversary can induce a slowdown in the commit rate. The vertical dashed lines represent the start and end of the attack.

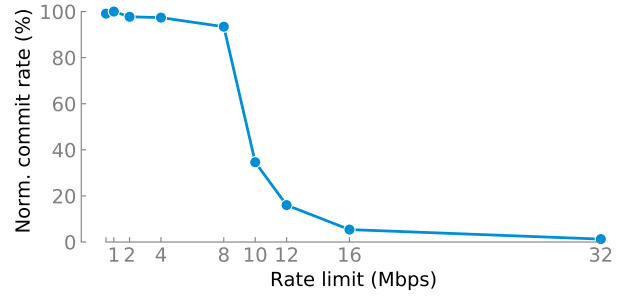


Figure 11: Finding a good rate-limiting threshold. With 8 validators and 8 attacking machines, the adversary needs more than 8 Mbps of aggregate attack traffic towards each target validator to achieve a loss of liveness.

Rate-limiting Threshold Effects. Setting an incorrect rate-limit for the ARL system may still allow adversaries to reduce the consensus protocol’s throughput. This is shown in Fig. 10a. With the higher (“loose” Fig. 10b) rate limit, the adversary can still force packet drops and slow down the commit rate.

Finding a good rate-limiting threshold. ?? shows an experiment to determine an appropriate value for the rate-limiting threshold as described in §8.