Does My BFT Protocol Implementation Scale?

Christian Berger
cb@sec.uni-passau.de
University of Passau
Passau, Germany

Sadok Ben Toumia
bentou01@ads.uni-passau.de
University of Passau
Passau, Germany

Hans P. Reiser
hansr@ru.is
Reykjavík University
Reykjavík, Iceland

Abstract

The novel blockchain generation of Byzantine fault-tolerant (BFT) state machine replication (SMR) protocols focuses on scalability and performance to meet the requirements of distributed ledger technology (DLT), e.g., decentralization and geographic dispersion. Validating scalability and performance of BFT protocol implementations requires careful evaluation. While experiments with real protocol deployments usually offer the best realism, they are costly and time-consuming. In this paper, we explore simulation of unmodified BFT protocol implementations as a method for cheap and rapid protocol evaluation: We can accurately forecast the performance of a BFT protocol while experimentally scaling its environment, i.e., by varying the number of nodes or geographic dispersion. Our approach is resource-friendly and preserves application-realism, since existing BFT frameworks can be simply plugged into the simulation engine without requiring code modifications or re-implementation.

1 Introduction

The current transition towards Web3 presents many challenges in terms of scalability and performance of distributed ledger technology (DLT). Proof-of-Work [23] is still widely used today, even if it is not environmentally sustainable and can often not meet the performance requirements of applications [7]. Consequently, coordination-based Byzantine fault-tolerant (BFT) state machine replication (SMR) algorithms experienced renewed research interest [3, 32] – resulting in many novel BFT protocols with a focus on improving scalability [9, 11, 24, 31, 36], or boosting performance under geographic dispersion [5, 8, 20, 30].

It is a challenging endeavour to reason about the performance and run-time behavior of these novel BFT protocols. Research papers describing BFT protocols often contain a thorough evaluation using large-scale deployments that are conducted on cloud platforms like AWS, where experiments deploy up to several hundred nodes (e.g., like in [9, 11, 20, 24, 36] and many more) to demonstrate a protocol's performance and scalability.

Evaluations using real protocol deployments usually offer the best realism, but are costly and time-consuming. Thus, a reasonable alternative for cheap and rapid validation of BFT protocol implementations without the need for cloud providers can be to rely on either emulation or simulation.

Emulation vs. Simulation. Emulation tries to duplicate the exact behavior of what is being emulated. A clear advantage of emulation is how it preserves realism: BFT protocols still operate in real time and use real kernel and network protocols. As examples serve Mininet [15, 19], which creates a realistic virtual network running real kernel, switch and application code on a single machine, or Kollaps [14], a decentralized and dynamic topology emulator.

In contrast to emulation, simulation decouples simulated time from real time and employs abstractions that help accelerate executions: Aspects of interest are captured through a model, which means the simulation only mimics the protocol's environment or its behavior. This has the advantage of easier experimental control, excellent reproducibility (i.e., deterministic protocol runs) and increased scalability when compared to emulation. As a potential drawback remains the question of application realism since the model may not fairly enough reflect reality. Examples of simulators include ns-3 [26] and Shadow [16], which are both discrete-event network simulators for Internet applications.

Evaluating BFT Protocols. BFTSim [29] is the first simulator that was developed for an eye-to-eye comparison of BFT protocols but it lacks the necessary scalability to be useful for the newer "blockchain generation" of BFT protocols (and apparently only up to $n = 32$ PBFT [10] replicas can be successfully simulated [34]). A more recent tool [34] allows for scalable simulation of BFT protocols but it unfortunately requires a complete re-implementation of the BFT protocol in JavaScript. It also cannot make predictions on system throughput. Kollaps [14] was used to reproduce AWS-deployed experiments with BFT-SMaRt [6] and WHEAT [30] but it is not sufficiently resource-friendly as it executes the real application code in real-time, thus requiring many physical machines to conduct large-scale experiments.

Research Questions & Contributions. In this paper, we explore simulation as a method to evaluate BFT protocol implementations, which leads us to the following two research questions:

R1 What are properties of an ideal performance evaluation tool for the "blockchain generation" of BFT protocols?
R2 Can simulations help us to reason about the behavior of real BFT protocol implementations at a larger scale?

	BFTSim [29]	BFT Simulator [34]	Kollaps [14]	ns-3 [26]	Mininet [15, 19]	Phantom [17]
application layer realism	✗	✗	✓	✗	✓	✓
realistic networking	✓	(*high level model*)	✓	✓	✓	✓
scalability	✗	✓	✓	✓	✗	✓
resource friendliness	✓	✓	✗	✓	✗	✓
Byzantine attacker	(*only bengin faults*)	✓	✗	✗	✗	✗

Table 1. Comparison of different emulators and simulators in the context of BFT protocol research.

Our contributions aim for supporting validations of novel BFT protocol implementations for their practical deployments in large-scale DLT systems. We summarize our main findings as follows:

- First, we compare existing simulators and emulators to analyze properties of an ideal evaluation tool in the context of BFT protocol research (Section 2). A key finding is that the state-of-the-art is deficient as there is no resource-friendly evaluation tool to predict the performance (i.e., latency *and* throughput) of BFT protocols at a larger scale.
- Further, we present a tool that automates large-scale simulations of unmodified BFT protocol implementations using the Phantom simulator [17] given a simple experimental description. For the first time, experiments with *existing BFT protocol implementations* can be effortlessly set up, configured and fed into a simulation engine (Sections 3 and 4).
- We discovered that we can faithfully forecast the performance of BFT protocols because performance eventually becomes network-bound at a larger scale. Our evaluations compare results obtained from simulations with measurements of real protocol deployments (Section 5).

2 Related Work & Background

BFTSim [29] was the first simulator tailored for traditional BFT protocols like PBFT [10] and Zyzzyva [18]. Since these protocols were intended for only small groups of replicas, the limited scalability of the simulator was at that time not an issue. However, it makes BFTSim impractical for the newer BFT protocols. BFTSim demands a BFT protocol to be modeled in the P2 language [21], which is somewhat error-prone when considering the complexity of, e.g., PBFT's view change or Zyzzyva's many corner cases. Although BFTSim allows the simulation of faults, it only considers non-malicious behavior and left the extension to more sophisticated attacks for future work. It provides realistic networking using ns-2, and is resource-friendly as it runs on a single machine.

Recently, Wang et al. [34] presented a BFT simulator that demonstrated resource-friendliness, high scalability, and comes with an *attacker module* which includes a pre-defined set of attacks (partitioning, adaptive, rushing). The simulator does not mimic real network protocols. Instead, it tries to capture network characteristics in a high-level model where messages can be delayed by some variable sampled from a (to be defined) Gaussian or Poisson distribution. Like BFTSim, it does not provide application layer realism and demands the re-implementation of a BFT protocol in JavaScript. A further drawback is that it cannot measure system throughput, and is thus not suited for reasoning about system performance.

Further, related work also includes stochastic modelling of BFT protocols [25] and validations of BFT protocols through unit test generation [2].

There are simulators which are dedicated to blockchain research, such as Shadow-Bitcoin [22], Bitcoin blockchain simulator [13], BlockSim [12], SimBlock [1], and ChainSim [33]. These tools mainly focus on building models that capture the characteristics of Proof-of-Work and thus cannot easily be adopted for BFT protocol research.

Further, there are tools to emulate or simulate distributed applications: Mininet [15, 19] and Kollaps [14] are emulators that allow creating realistic networks (running real Internet protocols) and real application code with time being synchronous with the wallclock. Naturally, both approaches provide a high degree of realism, which comes at the cost of being less resource-friendly. Mininet is not scalable, a problem which was addressed later by Maxinet [35], which allows Mininet-emulated networks to spawn over several physical machines. Kollaps is a scalable emulator, but also requires many physical machines for large-scale experiments. Moreover, ns-3 [26] is a resource-friendly and scalable network simulator, but it requires the development of an application model, and thus does not preserve application layer realism.

Phantom [17] uses a hybrid emulation/simulation architecture: It executes real applications as native OS processes, co-opting the processes into a network and kernel simulation, and thus can scale to large system sizes. Phantom preserves application layer realism as real BFT protocol implementations are executed. At the same time, it is resource-friendly and runs on a single machine. Through its hybrid architecture, Phantom resides in a sweet-spot between ns-3 (pure simulator) and Mininet (pure emulator): It still provides sufficient application realism for the execution of BFT protocols, but is more resource-friendly and scalable than the emulators are.

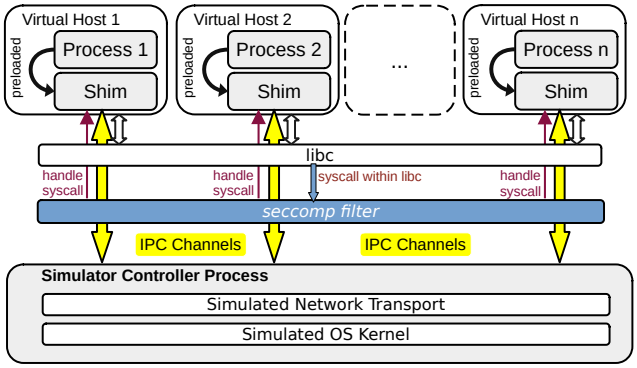


Figure 1. The Phantom architecture (high-level overview).

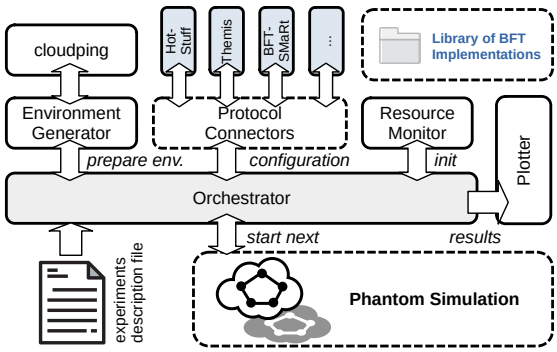


Figure 2. Our toolchain architecture for automating the setup of simulation runs of BFT protocols with Phantom.

As shown in Table 1, there is no perfect solution for simulating BFT protocols at scale yet. If we require both resource-friendliness and scalability, which we think are necessary characteristics to evaluate scalable BFT protocols in an inexpensive way, then only the BFT Simulator of Wang et al. [34] and Phantom [17] are viable options. Comparing these two, we decided to build our evaluation toolchain on top of Phantom, because it allows plug and play of BFT protocol implementations and can measure system throughput.

3 Preliminaries: Phantom

Phantom uses a hybrid simulation/emulation architecture, in which real, unmodified applications execute as normal processes on Linux and are hooked into the simulation through a system call interface using standard kernel facilities [17]. In Phantom, a network topology (the *environment*) can be described by specifying a graph, where *virtual hosts* are nodes and communication links are edges. The graph is attributed: For instance, virtual hosts specify available uplink/downlink bandwidth and links specify latency and packet loss.

Each virtual host can be used to run one or more applications. This results in the creation of real Linux processes that are initialized by the simulator controller process as managed processes (managed by a Phantom worker). The Phantom worker uses `LD_PRELOAD` to preload a shared library (called the *shim*) for co-opting its managed processes into the simulation (see Figure 1). `LD_PRELOAD` is extended by a second interception strategy, which uses `seccomp` for cases in which preloading does not work [17].

The shim constructs an inter-process communication channel (IPC) to the simulator controller process and intercepts functions at the system call interface. While the shim may directly emulate a few system calls, most system calls are forwarded and handled by the simulator controller process, which simulates kernel and networking functionality (for example the passage of time, I/O operations on `file`, `socket`, `pipe`, `timer`, event descriptors and packet transmissions) [17].

4 A Simulation Toolchain for BFT

Large-scale simulations of BFT protocols with Phantom require additional tooling support. This is mainly because of the following reasons: First, Phantom requires the generation of realistic and large network topologies for an arbitrary system size and the characteristics of their communication links should ideally resemble real-world deployments. This is crucial to allow realistic simulation of wide-area network environments.

Second, we need aid in setting up the BFT protocol implementations for their deployment, since bootstrapping a BFT protocol in Phantom involves many steps that can be tedious, error-prone, and protocol-specific. This means, for instance, the generation of protocol-specific run-time artifacts like cryptographic key material, or configuration files which differ for every BFT protocol.

Third, in the process of developing and testing BFT algorithms, different combinations of protocol settings result in numerous experiments being conducted. Since Phantom simulations run in virtual time, they can take hours, depending on the host system's specifications. For the sake of user experience and convenience, we find it is necessary for experiments to be specified in bulk and run sequentially without any need for user intervention.

Fourth, we may want to track and evaluate resources needed during simulation runs, such as CPU utilization and memory usage.

Fifth, when Phantom produces results, they reside in the file system and for convenience, we want to aggregate measurements of several simulations and map these to diagrams displaying to-be-specified metrics like throughput or latency. These reasons led us to develop Delphi-BFT[1], a tool on top of Phantom to simplify and accelerate the evaluation of unmodified BFT protocol implementations.

[1]Code open-source available at https://github.com/Delphi-BFT/tool.

4.1 Architecture

Delphi-BFT is composed of several components (see Figure 2) and follows a modular architecture, in that it is not tailored to a specific BFT protocol, but is easily extensible.

Orchestrator. The toolchain is administered by an orchestrator that manages all tools, i.e., for preparing an environment, configuring runtime artifacts for a BFT protocol, and initializing a resource monitor. The orchestrator invokes protocol connectors to set up a BFT protocol and loads *experiments description files* which contain a set of experiments to be conducted for the specified BFT protocol. Finally, the orchestrator starts Phantom, as soon as an experiment is ready for its execution.

Environment Generator. The environment generator creates network topologies as a complete graph for any system size. The network topologies resemble realistic deployment scenarios for a LAN or WAN setting. To create network graphs with network links reflecting a realistic geographic dispersion of nodes, the environment generator employs a cloudping component, which retrieves real round-trip latencies between all AWS regions from Cloudping[2]. This allows the tool to create network topologies which resemble real BFT protocol deployments on the AWS cloud infrastructure.

Protocol Connectors. For each BFT protocol implementation that we want to simulate, it is first necessary to create protocol configuration files and necessary keys. Since protocol options and cryptographic primitives can vary depending on the concrete BFT protocol, we implement the protocol-specific setup routine as a tool called *protocol connector*, which is invoked by the orchestrator. A connector must implement the methods `build()` and `configure()`. This way, it is simple to extend our toolchain and support new BFT protocols, as it only requires writing a new protocol connector (in our experience this means writing between 100 and 200 LoC).

Resource Monitor. The orchestrator initializes a resource monitor to collect information on resource consumption (like allocated memory and CPU time) during simulation runs and also the total simulation time. The user can use these statistics as indicators of a possible need for vertically scaling the host machine and as rough estimates for the necessary resources to run larger simulations.

Plotter. Results are stored on the file system by Phantom. They can be aggregated and mapped to specific diagrams for specifiable metrics like latency or throughput. For instance, it can create diagrams that display the performance of a BFT protocol for increasing system scale and aggregate several simulation runs for an increasing system size n (or any other variable).

[2]See https://www.cloudping.co/grid.

Table 2. BFT protocols that we employed for our evaluation.

framework	BFT protocol	language	repo on github.com
libhotstuff [36]	Hot-Stuff	C++	/hot-stuff/libhotstuff
themis [27]	PBFT	Rust	/ibr-ds/themis
bft-smart [6]	BFT-SMaRt	Java	/bft-smart/library

4.2 Configuration of Experiments

It is simple to support a new BFT protocol implementation by writing a new protocol connector: The connector defines how to build the implementation, configure it for deployment, and how to interpret the measurements taken from replica and client log files. Experiments are defined in *experiment description files* (EDF) that specify both environmental and protocol settings of an experiment divided in four categories:

- `misc`: general settings like duration of a simulation
- `network`: specifies bandwidth, latency, and packet loss. Latencies can be uniform (useful in a LAN) or for WAN experiments derived from the AWS latency map by specifying *locations* and the number of processes to be placed in each location like ['us-west-1': 1, 'eu-west-1': 1, 'sa-east-1':1, 'ap-southeast-2':1].
- `replica`: these are general or protocol-specific settings, e.g., number of replicas, batch size, or size of replies
- `client`: these are general or protocol-specific settings, e.g., number of clients, request size, or sending rate

The EDFs are the input of Delphi-BFT, which then automatically generates a sequence of corresponding Phantom experiment files and network topologies. In our appendix, we provide an example of an EDF that mimics one of the HotStuff experiments.

5 Results

In this section, we investigate on our second research question. We first use results from the HotStuff paper [36] to compare our simulation results with real measurements and reason about resource utilization of our simulations. Then, we show that we can achieve realistic results under geographic dispersion by simulating a WAN topology for BFT-SMaRt [6] replicas. Further, we experiment with a Rust-based PBFT implementation [27] to demonstrate compatibility with a third programming language (see Table 2).

5.1 HotStuff at Increasing System Scale

In our first evaluation, we mimic the evaluation setup of the HotStuff paper [36] to compare their measurements with our simulation results. The setup consists of more than a hundred virtual machines deployed in an AWS data center; each machine has up to 1.2 GB/s bandwidth and there is less than 1 ms latency between each pair of machines (we

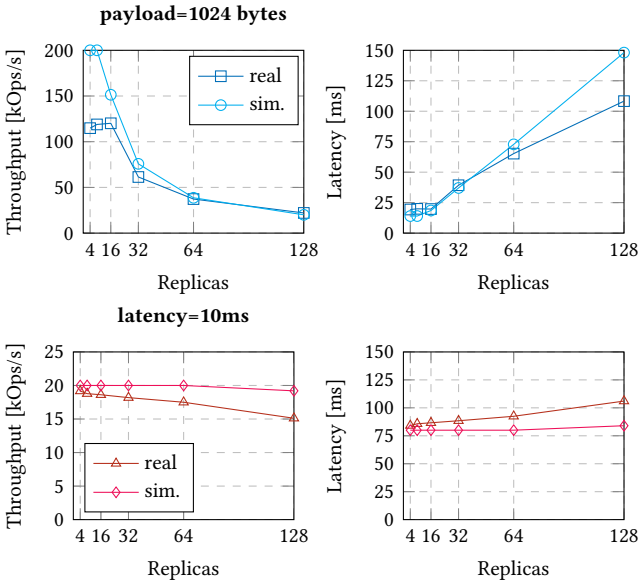


Figure 3. Performance of HotStuff and Simulated-HotStuff.

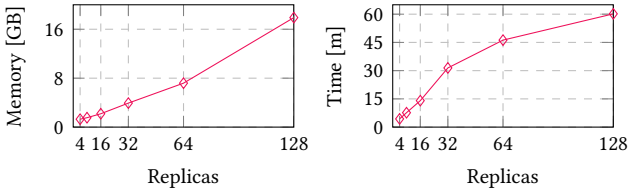
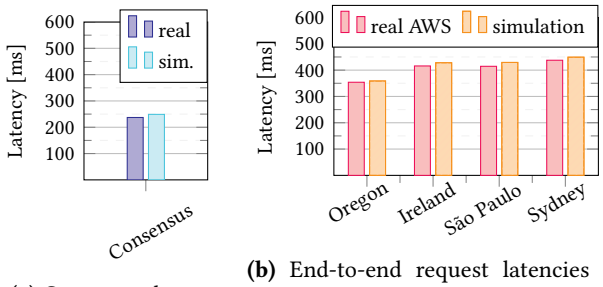


Figure 4. Resource consumption of simulations.

use 1 ms in the simulation). The employed batch size is 400. We compare against two measurement series: "1024" where the payload size of request and responses is 1024 bytes and "10ms" with empty payload, but the latency of all communication links is set to 10 ms. Our goal is to investigate how faithfully the performance of HotStuff can be predicted by regarding only the networking capabilities of replicas, which manifests at the point where the network becomes the bottleneck for system performance.

Observations. We display our results in Figure 3. The simulation results for the payload experiment indicate a similar trend as the real measurements, where performance starts to drop for $n \geq 32$. For a small-sized replica group, the network simulation predicts higher performance: 200k tx/s. This equals the theoretical maximum limited only through the 1 ms link latency which leads to pipelined HotStuff committing a batch of 400 requests every 2 ms. The difference in throughput decreases once the performance of HotStuff becomes more bandwidth-throttled (at $n \geq 32$). We also achieve close results in the "10ms" setting: 80 ms in the simulation vs 84.1 ms real, and 20k tx/s in the simulation vs. 19.2k tx/s real for $n = 4$; but with an increasing difference for higher n, i.e., 84 ms vs. 106 ms and 19k.2 tx/s vs. 15.1k tx/s for $n = 128$.



(a) Consensus latency. **(b)** End-to-end request latencies observed by clients in AWS regions.

Figure 5. Comparison of a real BFT-SMaRt WAN deployment on the AWS infrastructure with its simulated counterpart.

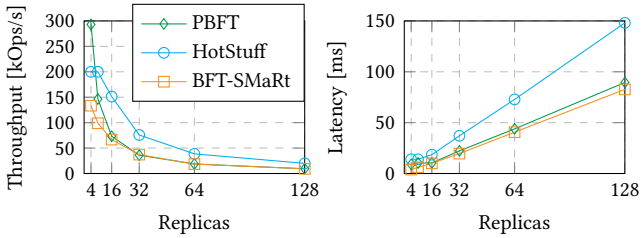


Figure 6. Simulation results of PBFT, BFT-SMaRt and HotStuff for 1 KiB request and response payload.

Resource Usage. Further, we investigate how resource utilization, i.e. memory usage and simulation time, grows with an increasing system scale. We run our HotStuff "10ms" simulations (which display a somewhat steady system performance for increasing system scale) on an Ubuntu 20.04 VM with 48 GB memory and 20 threads (16 threads used for simulation) on a host with an Intel Xeon Gold 6210U CPU at 2.5 GHz. We observe that active host memory and elapsed time grow with increasing system scale (see Fig. 4). We think it should be feasible to simulate up to 512 HotStuff replicas with a well-equipped host (with, e.g., 64 GB RAM).

5.2 BFT-SMaRt under Geographic Dispersion

Next, we experiment with geographic dispersion of BFT-SMaRt replicas, where each replica is located in a distinct AWS region. Our experimental setup is thus similar to experiments found in papers that research on latency improvements [4, 5, 30]. We employ a $n = 4$ configuration and choose the regions Oregon, Ireland, São Paulo and Sydney for the deployment of a replica and a client application each. We run clients one after another, and client each samples 1000 requests without payload and measures end-to-end latency, while the leader replica (in Oregon) measures the system's consensus latency. Further, we create an experiments description file mimicking this experiment and run it through our simulation toolchain to compare our simulation results with the real measurements.

Observations. We notice that consensus latency is slightly higher in the simulation (237 ms vs. 249 ms), and further, the simulation results also display slightly higher end-to-end request latencies in all clients (see Figure 5). The deviation between simulated and real execution is the lowest in Oregon (1.3%) and the highest in São Paulo (3.5%). A possible explanation of the deviation may be the high jitter of wide-area networks that is not fairly enough captured by the underlying network simulator. While we believe the degree of realism is good enough for the purpose of rapid evaluation of new BFT protocol implementations, further improvements should be considered as part of the future work.

5.3 PBFT at Increasing System Scale

In this experiment, we run simulations with 1KiB request and response payload with Themis [27] (a Rust-based implementation of PBFT) for an increasing system scale to compare the results against our HotStuff and BFT-SMaRt simulation results.

Observations. PBFT initially outperforms HotStuff, but then its throughput decreases more swiftly (as can be seen in the sharper curve in Figure 6). At $n = 128$, PBFT achieves up to 9.3k tx/s while HotStuff achieves up to 20k tx/s. Here, BFT-SMaRt, which employs the same agreement pattern as PBFT also achieves up to 9.2k tx/s throughput at similar latency values.

6 Future Work

Extending Evaluations. For future work, we intend to extend our evaluations to more BFT protocols, in particular, to evaluate the effectiveness of different communication strategies, like Gosig [20] (gossip) or Kauri [24] (tree-based) and compare them with the results obtained from Hot-Stuff (star-based) and PBFT (clique). In particular, we can explore the performance of these protocols under different network characteristics and for an increasing system scale. A high-level simulation model previously studied the effect of different message exchange patterns of BFT protocols [28] but it lacks applicability for reasoning about real system metrics.

CPU Model. We think a CPU model could improve simulation results for evaluations of either (1) small sized replica groups or (2) experiments with empty payload – in both cases the CPU may be the dominating factor and not the network. Up to now, we use Phantom only as a network simulator and all computations, such as creating or verifying signatures, take no time. It might be possible to capture most of the computational work by only modeling a few methods, in particular, the cryptographic primitives (like in BFTSim [29]). Currently, Phantom plans the introduction of a CPU model as a future milestone for development and we will try to utilize it to improve our simulation results.

Attacker Model. Moreover, we have in view to introduce an attacker model to reason about the impact of attacks on system performance. For this reason, we seek inspiration from the Twins [2] methodology, a recent approach for validating BFT protocols: Twins is a unit test case generator that can simulate Byzantine attacks by duplicating cryptographic identities of replicas (which then leads to forgotten protocol states, or equivocations) and it can be quite useful in a simulator to explore a variety of attacking scenarios.

Acknowledgments

This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) grant number 446811880 (BFT2Chain). We are thankful for the help we received from the active and responsive Phantom development team on github.

References

[1] Yusuke Aoki, Kai Otsuki, Takeshi Kaneko, Ryohei Banno, and Kazuyuki Shudo. 2019. SimBlock: A blockchain network simulator. In *IEEE Conf. on Computer Communications Workshops*. IEEE Comp. Soc., Washington, DC, USA, 325–329.

[2] Shehar Bano, Alberto Sonnino, Andrey Chursin, Dmitri Perelman, Zekun Li, Avery Ching, and Dahlia Malkhi. 2022. Twins: BFT Systems Made Robust. In *25th Int. Conf. on Principles of Distributed Systems*, Vol. 217. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 7:1–7:29.

[3] Christian Berger and Hans P. Reiser. 2018. Scaling Byzantine Consensus: A Broad Analysis. In *the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*. ACM, New York, NY, 13–18.

[4] Christian Berger, Hans P. Reiser, and Alysson Bessani. 2021. Making Reads in BFT State Machine Replication Fast, Linearizable, and Live. In *40th Int. Symp. on Reliable Distributed Systems*. IEEE Comp. Soc., Washington, DC, USA, 1–12.

[5] Christian Berger, Hans P. Reiser, João Sousa, and Alysson Neves Bessani. 2022. AWARE: Adaptive wide-area replication for fast and resilient Byzantine consensus. *IEEE Trans. on Dependable and Secure Computing* 19, 3 (2022), 1605–1620.

[6] Alysson Bessani, João Sousa, and Eduardo EP Alchieri. 2014. State machine replication for the masses with BFT-SMaRt. In *44th Annu. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*. IEEE Comp. Soc., Washington, DC, USA, 355–362.

[7] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. 2015. SoK: Research perspectives and challenges for bitcoin and cryptocurrencies. In *IEEE Symp. on security and privacy*. IEEE Comp. Soc., Washington, DC, USA, 104–121.

[8] Loïck Bonniot, Christoph Neumann, and François Taïani. 2020. PnyxDB: a lightweight leaderless democratic Byzantine fault tolerant replicated datastore. In *Int. Symp. on Reliable Distributed Systems*. IEEE Comp. Soc., Washington, DC, USA, 155–164.

[9] Daniel Cason, Enrique Fynn, Nenad Milosevic, Zarko Milosevic, Ethan Buchman, and Fernando Pedone. 2021. The design, architecture and performance of the Tendermint Blockchain Network. In *40th Int. Symp. on Reliable Distributed Systems*. IEEE Comp. Soc., Washington, DC, USA, 23–33.

[10] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *Proc. of the third symposium on operating systems design and implementation (OSDI)*. USENIX Association, Berkeley, CA, USA, 173–186.

[11] Tyler Crain, Christopher Natoli, and Vincent Gramoli. 2021. Red belly: A secure, fair and scalable open blockchain. In *IEEE Symp. on Security and Privacy*. IEEE Comp. Soc., Washington, DC, USA, 466–483.
[12] Carlos Faria and Miguel Correia. 2019. BlockSim: blockchain simulator. In *IEEE Int. Conf. on Blockchain*. IEEE Comp. Soc., Washington, DC, USA, 439–446.
[13] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. On the security and performance of proof of work blockchains. In *ACM SIGSAC CCS*. ACM, New York, NY, 3–16.
[14] Paulo Gouveia, João Neves, Carlos Segarra, Luca Liechti, Shady Issa, Valerio Schiavoni, and Miguel Matos. 2020. Kollaps: decentralized and dynamic topology emulation. In *15th EuroSys Conf.* ACM, New York, NY, 1–16.
[15] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. 2012. Reproducible network experiments using container-based emulation. In *8th Int. Conf. on Emerging networking experiments and technologies*. ACM, New York, NY, 253–264.
[16] Rob Jansen and Nicholas Hopper. 2012. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *19th Annu. NDSS*. The Internet Society, 18 pages.
[17] Rob Jansen, Jim Newsome, and Ryan Wails. 2022. Co-opting Linux Processes for High-Performance Network Simulation. In *USENIX ATC 22*. USENIX Association, Berkeley, CA, USA, 327–350.
[18] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. 2007. Zyzzyva: speculative Byzantine fault tolerance. In *the 21st ACM SIGOPS SOSP*. ACM, New York, NY, 45–58.
[19] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A network in a laptop: rapid prototyping for software-defined networks. In *9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, New York, NY, 1–6.
[20] Peilun Li, Guosai Wang, Xiaoqi Chen, Fan Long, and Wei Xu. 2020. Gosig: a scalable and high-performance byzantine consensus for consortium blockchains. In *11th ACM Symp. on Cloud Computing*. ACM, New York, NY, 223–237.
[21] Boon Thau Loo, Tyson Condie, Joseph M. Hellerstein, Petros Maniatis, Timothy Roscoe, and Ion Stoica. 2005. Implementing declarative overlays. In *12th ACM SOSP*. ACM, New York, NY, 75–90.
[22] Andrew Miller and Rob Jansen. 2015. Shadow-Bitcoin: Scalable Simulation via Direct Execution of Multi-Threaded Applications. In *8th Workshop on Cyber Security Experimentation and Test*. USENIX Association, Berkeley, CA, USA, 9 pages.
[23] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf
[24] Ray Neiheiser, Miguel Matos, and Luís Rodrigues. 2021. Kauri: Scalable BFT consensus with pipelined tree-based dissemination and aggregation. In *ACM SIGOPS 28th SOSP*. ACM, New York, NY, 35–48.
[25] Martin Nischwitz, Marko Esche, and Florian Tschorsch. 2021. Bernoulli meets PBFT: Modeling BFT protocols in the presence of dynamic failures. In *16th Conference on Computer Science and Intelligence Systems*. IEEE Comp. Soc., Washington, DC, USA, 291–300.
[26] George F Riley and Thomas R Henderson. 2010. The ns-3 network simulator. In *Modeling and tools for network simulation*. Springer, Cham, 15–34.
[27] Signe Rüsch, Kai Bleeke, and Rüdiger Kapitza. 2019. Themis: An Efficient and Memory-Safe BFT Framework in Rust: Research Statement. In *3rd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*. ACM, NY, 9–10.
[28] Fábio Silva, Ana Alonso, José Pereira, and Rui Oliveira. 2020. A comparison of message exchange patterns in BFT protocols. In *IFIP Int. Conf. on Distributed Applications and Interoperable Systems*. Springer, Cham, 104–120.
[29] Atul Singh, Tathagata Das, Petros Maniatis, Peter Druschel, and Timothy Roscoe. 2008. BFT Protocols Under Fire.. In *NSDI*, Vol. 8. USENIX Association, Berkeley, CA, USA, 189–204.
[30] João Sousa and Alysson Bessani. 2015. Separating the WHEAT from the Chaff: An Empirical Design for Geo-Replicated State Machines. In *34th IEEE Symp. on Reliable Distributed Systems*. IEEE Comp. Soc., Washington, DC, USA, 146–155.
[31] Chrysoula Stathakopoulou, Tudor David, Matej Pavlovic, and Marko Vukolić. 2021. Mir-BFT: High-Throughput Robust BFT for Decentralized Networks. arXiv:1906.05552 [cs.DC]
[32] Marko Vukolić. 2015. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *Int. Workshop on Open Problems in Network Security*. Springer, Cham, 112–125.
[33] Bozhi Wang, Shiping Chen, Lina Yao, and Qin Wang. 2020. ChainSim: A P2P Blockchain Simulation Framework. In *CCF China Blockchain Conf.* Springer, Singapore, 1–16.
[34] Ping-Lun Wang, Tzu-Wei Chao, Chia-Chien Wu, and Hsu-Chun Hsiao. 2022. Tool: An Efficient and Flexible Simulator for Byzantine Fault-Tolerant Protocols. In *52th Annu. IEEE/IFIP Int. Conf. on Dependable Systems and Networks*. IEEE Comp. Soc., Washington, DC, USA, 287–294.
[35] Philip Wette, Martin Dräxler, Arne Schwabe, Felix Wallaschek, Mohammad Hassan Zahraee, and Holger Karl. 2014. Maxinet: Distributed emulation of software-defined networks. In *IFIP Networking Conf.* IEEE Comp. Soc., Washington, DC, USA, 1–9.
[36] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2018. HotStuff: BFT consensus in the lens of blockchain. https://doi.org/10.48550/arXiv.1803.05069 arXiv:1803.05069 [cs.DC]

Appendix

```yaml
protocolName: hotstuff
protocolConnectorPath: ./connectors/hotstuff.js
experiments:
  - 128rep:
      misc:
        duration: 30 s
      network:
        bandwidthUp: 10 Gibits
        bandwidthDown: 10 Gibits
        latency:
          uniform: true
          replicas: 1000 us
          clients: 1000 us
        packetLoss: 0.0
      replica:
        replicas: 128
        blockSize: 400
        replySize: 1024
      client:
        clients: 16
        numberOfHosts: 2
        outStandingPerClient: 175
        requestSize: 1024
```

Figure 7. Example of an experiment description file.