Secure DataCapsule Replication Using Blockchain Byzantine Protocols
Sisha Rao, Stephen Jayakar, Lucie Choi
{sishaar, stephenjayakar, lucie0407}@berkeley.edu

Introduction
Modern applications employ data replication over geographical boundaries in order to increase data availability and bolster system reliability. A fundamental characteristic of distributed system reliability is a tolerance for failure within one of its components. Byzantine Fault Tolerance (BFT) is one popular approach for introducing resiliency in the face of such component failures, and modern BFT protocols have introduced optimizations to scale the size of fault-tolerant networks to be in the hundreds of nodes. The DataCapsule is a cryptographically-hardened log data structure stored on an infrastructure denoted as the Global Data Plane (GDP). DataCapsule CRUD operations are routed within the GDP network which replaces physical addresses (e.g. IP) with a flat address space. This provides applications a straightforward means of distributing and reorganizing DataCapsules across geographical boundaries.

We have built a Proof-of-Concept demonstrating Byzantine Fault Tolerance with replicas represented as DataCapsules, and have gathered measurements of the system’s performance in a variety of configurations, indicating to us that providing efficient fault-tolerance within the GDP network is possible. As the envisioned fundamental unit of data in a federated edge-computing network, introducing fault-tolerance atop a GDP network demonstrates the feasibility of DataCapsules as the underlying data structure for use cases involving data replication.

Notable Benchmarks
Our implementation was built using a Docker container porting the Ubuntu Linux kernel with an x86 64-bit architecture and a GNU/Linux OS onto Macintosh OS X. The container was configured to run a Quad-core Intel i5-4258U CPU @ 2.40GHz with 2.0 GB of memory and 1.0 GB of swap memory. The Concord-BFT engine limits a maximum of 3 replicas per core, as such we were limited to testing with at most 12 replicas. Figure 3 shows the runtimes of various configurations of SBFT. The Concord-BFT engine blocks incoming commits for the completion of current commits in order to preserve serializability. Thus we observe that as the number of clients increases, the runtime per commit increases. Figure 4 shows runtimes across a range of increasing network latencies with one configuration that stores state in memory compared against another configuration using DataCapsules. We observe that DataCapsules perform similar to the control with the additional latency for callouts to GDP, whereas for shorter latencies, the slow path commit protocol within the Concord engine is called. We conclude fine-tuning of timeout parameters is needed.

Implementation with the Concord-BFT Engine
We employed the Concord-BFT engine, an implementation of the Scaling BFT (SBFT) protocol built by VmWare. As the name suggests, SBFT is optimized for a network with hundreds of geo-distributed replicas. SBFT makes 4 key additions to PBFT, (1) going from PBFT to Linear PBFT, (2) adding a fast path, (3) employing threshold signatures for acknowledgement, and (4) low replica latency to improve resilience and performance. The commit algorithm is similar to a standard 2PC protocol, with the intermediary stages corresponding to cryptographic verification of messages from replicas.

Our contributions to developing the Proof-of-Concept atop the GDP infrastructure was the creation of a DataCapsule API wrapper as well as defining Replicas whose execution directs reads and writes through the wrapper. The API wrapper defines functionality for the interaction with DataCapsules, such as various naming conventions for replicas, formatting of reads and writes, etc. The Concord engine supports configurations with multiple clients, multiple replicas (both faulty and non-faulty) as well as slow to respond replicas. The SBFT system has a dual mode view change protocol when slow replicas are present in the network, however as this is a feature unique to SBFT and not to BFT as a whole, we omitted configurations including slow replicas in our metrics.

References and Acknowledgements
A special thanks to Professor Kubiatowicz for his continued support and guidance throughout the course of this project, as well as Igor Golikov and the Concord-BFT team from VMWare for their help with understanding the Concord-BFT engine.

(4)https://hub.docker.com/_/ubuntu