Orchestrating a brighter world

NEC

Open Source Summit Japan 2019

MinBFT, Hyperledger Lab

Open Source Project to Develop Efficient Consensus Protocol

Yuta Namiki

NEC Corporation

\Orchestrating a brighter world

NEC brings together and integrates technology and expertise to create the ICT-enabled society of tomorrow.
We collaborate closely with partners and customers around the world, orchestrating each project to ensure all its parts are fine-tuned to local needs.

Every day, our innovative solutions for society contribute to greater safety, security, efficiency and equality, and enable people to live brighter lives.

Agenda

Introduction

Blockchain and Consensus Protocol

MinBFT Protocol

Approach

Implementation

Introduction

- NEC involved in R&D of blockchain technologies since 2012
- This project originates from NEC’s internal research initiative
- To discover efficient practical consensus algorithms
- To raise awareness about the benefits of TEE for BFT protocols

Hyperledger Labs

\Orchestrating a brighter world NEC

Hyperledger

- Project hosted by The Linux Foundation
- Hyperledger is an open source collaborative effort created to advance cross-industry blockchain technologies
- It is a global collaboration, including leaders in finance, banking, Internet of Things, supply chains, manufacturing and Technology.

https://www.hyperledger.org/

Hyperledger Labs

- A space for innovation and testing of ideas.
- Hyperledger Labs will allow teams to experiment with new frameworks or new modules without the promise of stable code or MVP

- We have launched MinBFT lab here in August 2018
- https://github.com/hyperledger-labs/minbft

https://wiki.hyperledger.org/display/labs

Blockchain and Consensus Protocol

\Orchestrating a brighter world NEC

Blockchain

Bitcoin (2009)

- A cryptocurrency
- Decentralized: peer-to-peer, without a trusted authority or central server

Blockchain

- Introduced as a ledger (a distributed database) in Bitcoin
- Generalized not only for cryptocurrencies
- Many (open source) implementations
  - Fabric, Sawtooth, Iroha (these are in Hyperledger project [1]) , Ethereum [2] etc.

https://en.wikipedia.org/wiki/Bitcoin, https://en.wikipedia.org/wiki/Blockchain
[1] https://www.hyperledger.org/, [2] https://www.ethereum.org/

Blockchain and Consensus Protocol

Blockchain involves a consensus protocol

Role in Blockchain

- Make an agreement
  - In Bitcoin, agreement on ownership of a coin to prevent “double-spending”
  - In general, agreement on order of transactions, result, ...

Requirements in Blockchain

- Tolerance for Byzantine failure

Byzantine Failure

Failure models

- Crash failure
  - Stops responding
- Byzantine failure
  - Arbitrary behavior
    - Responds with a lie, etc.
  - Hard to tolerate

Behavior of a malicious attacker in a Blockchain network is modeled as Byzantine failure

- Blockchain relies on a Byzantine fault tolerance (BFT) protocol

BFT Consensus Protocol in Blockchain

Proof of Work (PoW)

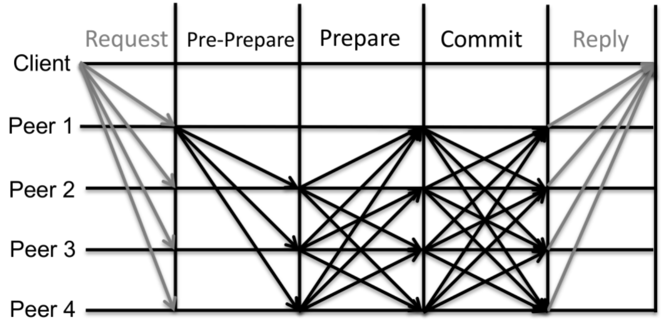
- Used in Bitcoin
- Suitable for public blockchains (“open” network like Bitcoin): scalable to thousands of nodes, but no finality

Practical Byzantine Fault Tolerance (PBFT)

- Used in Hyperledger Fabric (version 0.6)
- Suitable for private blockchains: (relatively) high performance and finality, but limited scalability
- Focus on this in my presentation

PBFT: Protocol

- 0. A node is elected as the primary node in the network to accept client request
- 1. The primary node multicasts PREPREPARE messages to the other nodes, in which a sequence number is also assigned to the message along with a signature
- 2. Upon reception of the PREPREPARE message, each nodes validates the signature
- 3. They multicast a PREPARE message that confirms the PREPREPARE message
- 4. A node receives $2f$ PREPARE messages that match the PRE-PREPARE message, multicasts a COMMIT message
- 5. A node receives $2f + 1$ COMMIT messages that the PREPARE message, it executes the request



MinBFT

\Orchestrating a brighter world NEC

Road to Efficient Byzantine Fault Tolerance (MinBFT)

Limitations in PBFT

- Performance
- Scalability: scales to few tens of nodes

MinBFT proposed by G. S. Veronese et al. in 2013

- Leverage a secure hardware to make the protocol efficient so that overcome the limitations

Intel Software Guard Extensions (SGX)

- An implementation of the "secure hardware" or Trusted Execution Environment (TEE)
- Introduced with 6th generation Intel Processors (Skylake) in 2015
- Provides an isolated environment (“enclave”) for storing sensitive data and running codes
  - Preserve confidentiality and integrity of the data and code
  - Even software running at higher privilege levels can neither access nor modify the data

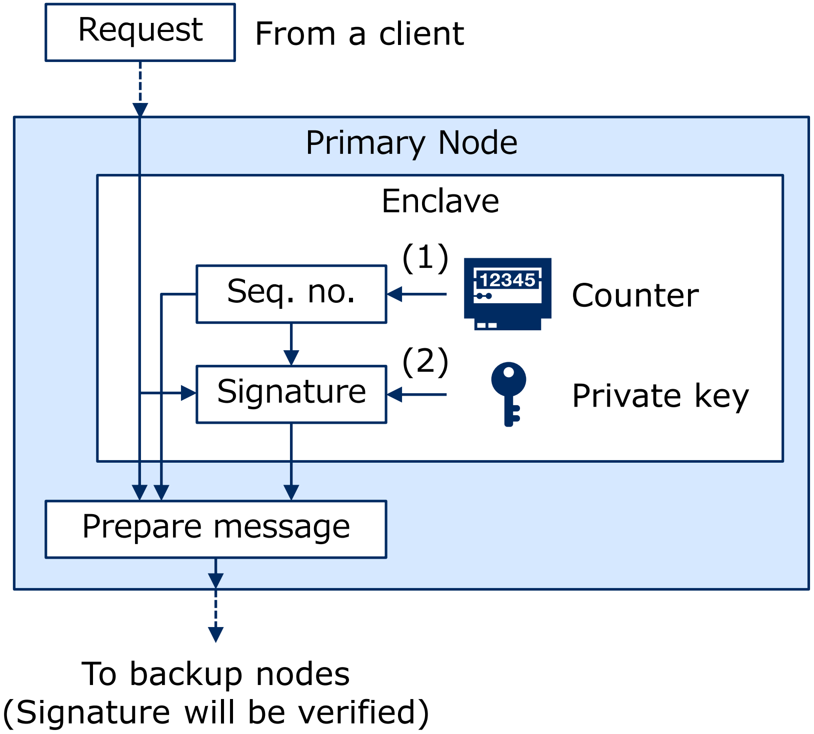
Intel SGX in MinBFT: Contents of Enclave

Data

- A monotonic counter
- A private key

Code

- (1) Assigning a sequence number to a message
  - Increment the counter and assign its value to a request in a primary node
- (2) Signing a message
  - Create a prepare message with the sequence number and a signature by a private key in the enclave



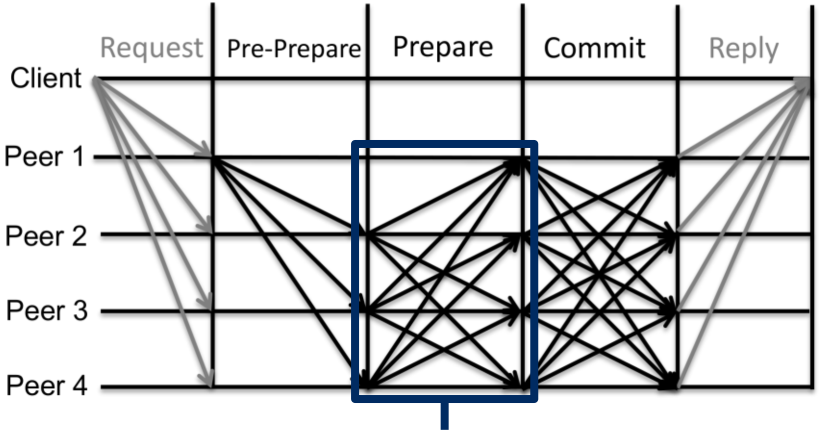
Intel SGX in MinBFT: Effects (1)

Guarantees by Intel SGX

- A primary never assign a sequence number *n* to different requests *m1* and *m2*
  - A counter for sequence numbers is in an enclave; it cannot be interfered
- A sequence number *n* assigned to a request *m*, which is described in a prepare message is issued by a primary
  - The prepare message is signed and the signing is processed in an enclave (the private key is protected)
- All backups receive the same combination <*m*, *n*> (prevent equivocation)

Intel SGX in MinBFT: Effects (2)

- Backup nodes can trust a sequence number assigned by a primary node
  - In PBFT, all backup nodes needs to confirm everyone received the same sequence number by broadcasting a message in prepare phase
- MinBFT can omit the phase under the above condition



Confirmation of a sequence number (PBFT)

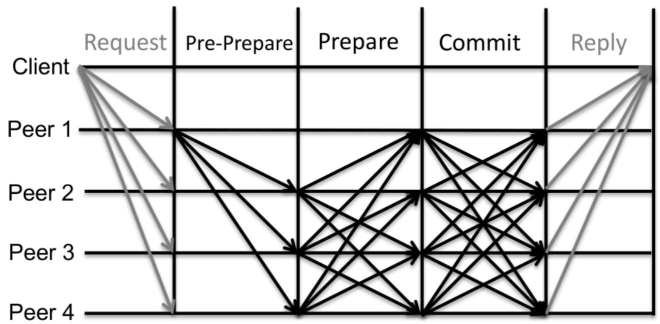
Intel SGX in MinBFT: Effects (3)

Reduction of total nodes

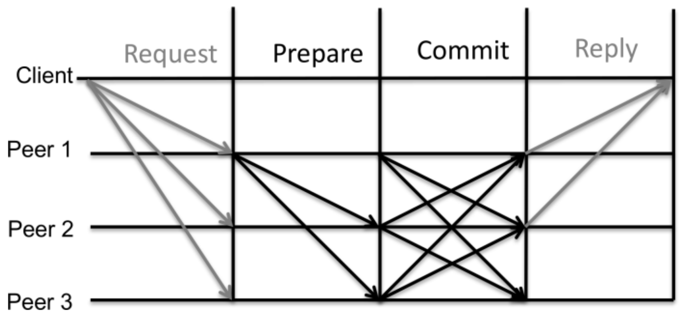
- MinBFT requires $2f + 1$ nodes to tolerate f faulty while PBFT requires $3f + 1$ nodes
  - f nodes could be faulty; the system needs to proceed with $(n - f)$ replies
  - The $(n - f)$ replies could contain up to f malicious replies
  - Therefore a node expect at most $((n - f) - f) = n - 2f$ correct replies
  - In PBFT, this must be majority: $n - 2f > f$ therefore $n >= 3f + 1$
  - In MinBFT, this must be at least one (Intel SGX guarantees every nodes receive same requests): $n - 2f >= 1$ therefore $n >= 2f + 1$

MinBFT: Protocol

- 0. A node is elected as the primary node in the network to accept client request
- 1. The primary node broadcasts PREPARE messages to the other nodes, in which a sequence number is also assigned to the message along with a signature by the secure hardware
- 2. Upon reception of the PREPARE message, each nodes validates the signature as well as the sequence number to see if it is incremental
- 3. They broadcast a COMMIT message that confirms the PREPARE message
- 4. A node receives $f + 1$ COMMIT message, it executes the request



PBFT



EBFT

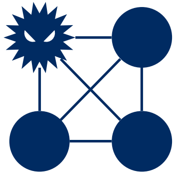
PBFT vs. MinBFT

Comparison

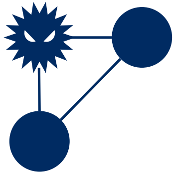
- MinBFT can reduce total number of nodes and communication rounds by leveraging a secure hardware
- The efficiency increases system throughput

	PBFT	EBFT
Secure Hardware	No	Yes
Total Nodes	$3f + 1$	$2f + 1$
Communication Rounds	3	2

Total Nodes



PBFT: 4 (for $f = 1$)



EBFT: 3

Communication Rounds

Pre-prepare → Prepare → Commit

PBFT: 3

Prepare → Commit

EBFT: 2

Approach

[Objectives]
[Scope]
[Status]

Approach

Objectives

- **Pluggable software component**
- **Clean design**
  - Modularity
  - Testability
- **Easy integration**
  - Abstract interfaces
  - Simple blockchain-like example
- **Best practices**
  - Embedded documentation
  - Unit and integration tests
  - Automatic code quality check (linting)
  - Continuous integration

Approach

Scope

Core components

- Core MinBFT protocol
- USIG service
- Client functionality

External components (sample code)

- Authentication
- Network connectivity
- Configuration
- CLI application to run a replica/client instance

Approach

Status

- Experimental development stage
- Hyperledger Lab
- Features implemented
  - Normal case operation
  - SGX USIG
  - Simple blockchain-like example
- Features considered
  - *View change operation (under development)*
  - Garbage collection and checkpoints
  - USIG enclave attestation
  - Faulty node recovery
  - Documentation improvement
  - Testing improvement

Implementation

[Details](#)

[Structure](#)

[USIG](#)

Implementation

Details

Language

- Most of the code in Go
- SGX enclave in C

Core dependencies

- Go v1.11
- golang/protobuf v1.1
- Intel® SGX SDK for Linux v2.4
- Tested on Ubuntu 18.04 LTS

Licence

- Source code under Apache Licence 2.0
- Documentation under Creative Commons Attribution 4.0 International License

Implementation

Structure

- **`api`** – definition of API between core and external components
- **`client`** – implementation of client-side part of the protocol
- **`core`** – implementation of core consensus protocol
- **`usig`** – implementation of USIG, tamper-proof component
- **`messages`** – definition of the protocol messages
- **`sample`** – sample implementation of external interfaces
  - **`authentication`** - generation and verification of authentication tags
    - **`keytool`** – tool to generate sample key set file
  - **`net`** – network connectivity
  - **`config`** – consensus configuration provider
  - **`requestconsumer`** – service executing ordered requests
  - **`peer`** – CLI application to run a replica/client instance

Implementation

USIG

- Intel ® SGX enclave as tamperproof component
- USIG-Sign using ECDSA (FIPS 186-3) signature
- Private key generated inside enclave and protected by SGX
- Key pair can be sealed and stored permanently
- Remote attestation can be decoupled from consensus instantiation
- No dependency on SGX Monotonic Counter (Trusted Platform Service)
- Ephemeral counter value
- Ephemeral unique epoch value for each enclave instance
- USIG identity as key pair combined with epoch value

Contributing

- Any feedback highly appreciated
  - Questions
  - Suggestions
  - Bug reports
  - Change requests etc.
- GitHub issue or pull request, as appropriate

References

- Hyperledger Lab: https://github.com/hyperledger-labs/minbft
- Efficient BFT Paper: https://goo.gl/oQs43M
- Efficient BFT Proof: https://goo.gl/tMxSrs
- NEC Activities on Blockchain: https://goo.gl/aDGyZM
- NEC Laboratories Europe GmbH: http://www.neclab.eu/

\Orchestrating a brighter world

NEC