BLOXY: Providing Transparent and Generic BFT-Based Ordering Services for Blockchains

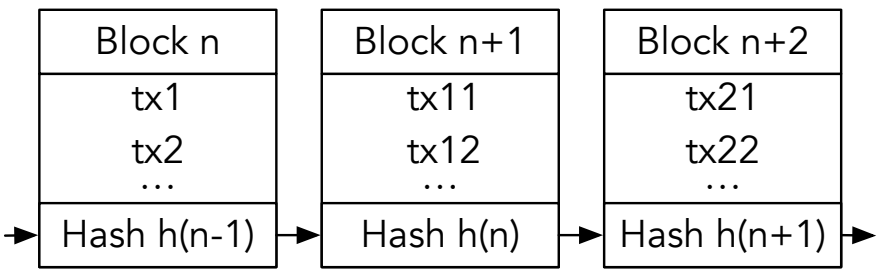
Symposium on Reliable Distributed Systems, October 4, 2019

Signe Rüsch, Kai Bleeke, Rüdiger Kapitza
ruesch@ibr.cs.tu-bs.de
Technische Universität Braunschweig, Germany

Funded by DFG

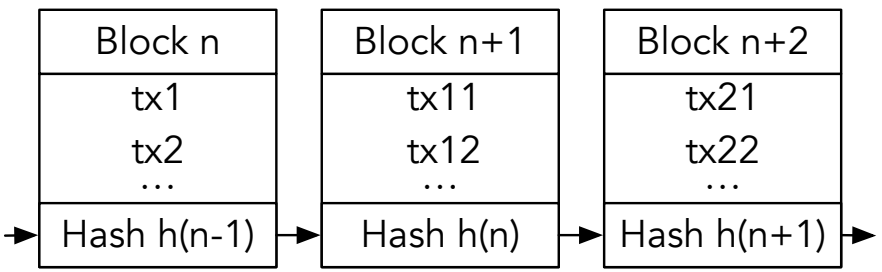
Blockchain Technology for Industry Usage

- Transactions are stored decentralised and tamper-proof
- **Permissioned** blockchains:
  - Number of users and access to data regulated
  - Interesting for companies and organisations
  - Several use cases: supply chain management, distributed storage, ...



Blockchain Technology for Industry Usage

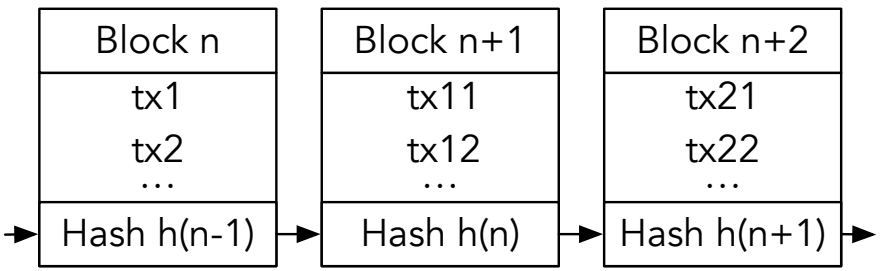
- Transactions are stored decentralised and tamper-proof
- **Permissioned** blockchains:
  - Number of users and access to data regulated
  - Interesting for companies and organisations
  - Several use cases: supply chain management, distributed storage, ...
- Byzantine fault tolerant (**BFT**) protocols:
  - Consensus even with participants showing arbitrarily wrong behaviour
  - Suitable for block creation



Blockchain Technology for Industry Usage

- Transactions are stored decentralised and tamper-proof
- **Permissioned** blockchains:
  - Number of users and access to data regulated
  - Interesting for companies and organisations
  - Several use cases: supply chain management, distributed storage, ...
- Byzantine fault tolerant (**BFT**) protocols:
  - Consensus even with participants showing arbitrarily wrong behaviour
  - Suitable for block creation

↳ Different use cases lead to different **requirements** for consensus!

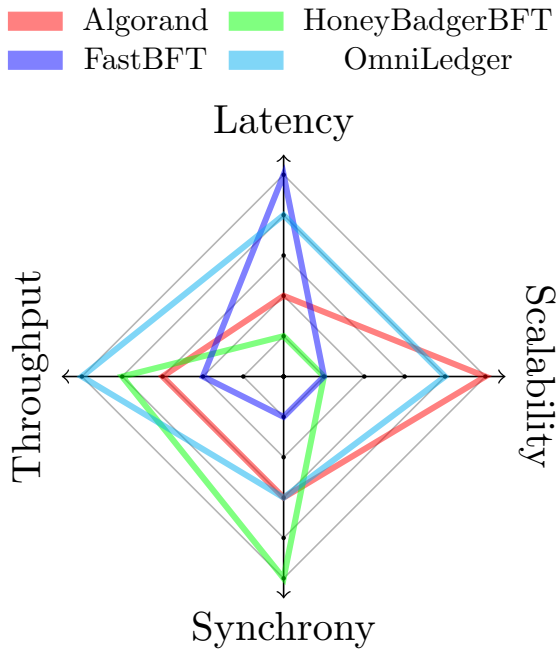


BFT Protocols

- **Common** structure:
  - Client sends request
  - Replicas order and execute request
  - Client performs **majority voting** on replies

BFT Protocols

- **Common** structure:
  - Client sends request
  - Replicas order and execute request
  - Client performs **majority voting** on replies
- **Optimized** for different aspects:
  - Scalability: Algorand [1]
  - Throughput: OmniLedger [2]
  - Latency: FastBFT [3]
  - (A)Synchrony: HoneyBadgerBFT [4]

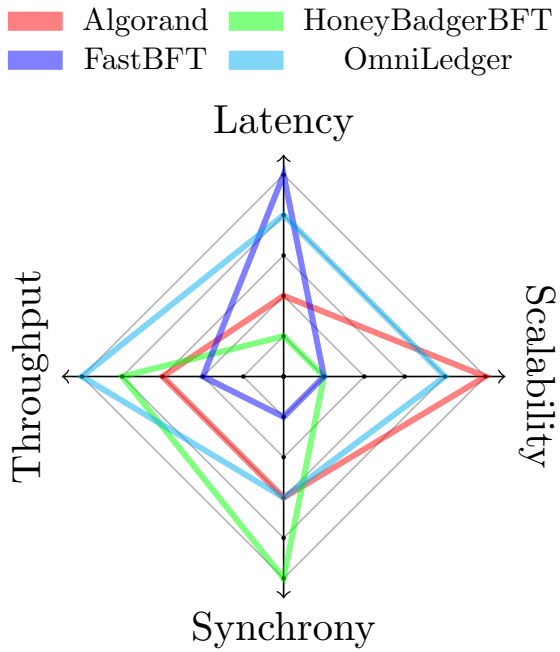


[1] Gilad et al., SOSP'17
[2] Kokoris-Kogias et al., IEEE S&P'18
[3] Liu et al., 2016
[4] Miller et al., CCS'16

BFT Protocols

- **Common** structure:
  - Client sends request
  - Replicas order and execute request
  - Client performs **majority voting** on replies
- **Optimized** for different aspects:
  - Scalability: Algorand [1]
  - Throughput: OmniLedger [2]
  - Latency: FastBFT [3]
  - (A)Synchrony: HoneyBadgerBFT [4]

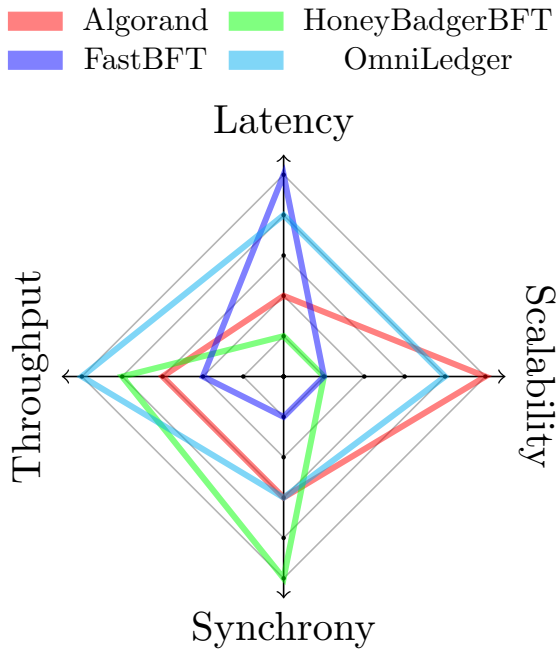
↪ No **one-size-fits-all** BFT solution!



[1] Gilad et al., SOSP'17
[2] Kokoris-Kogias et al., IEEE S&P'18
[3] Liu et al., 2016
[4] Miller et al., CCS'16

BFT Protocols

- **Common** structure:
  - Client sends request
  - Replicas order and execute request
  - Client performs **majority voting** on replies
- **Optimized** for different aspects:
  - Scalability: Algorand [1]
  - Throughput: OmniLedger [2]
  - Latency: FastBFT [3]
  - (A)Synchrony: HoneyBadgerBFT [4]



[1] Gilad et al., SOSP'17
[2] Kokoris-Kogias et al., IEEE S&P'18
[3] Liu et al., 2016
[4] Miller et al., CCS'16

↳ No **one-size-fits-all** BFT solution!

↳ Allow operators to **choose protocol fitting to use case!**

Hyperledger Fabric

- Permissioned blockchain platform [Androulaki et al., EuroSys'18]
- Modular design with **pluggable consensus**

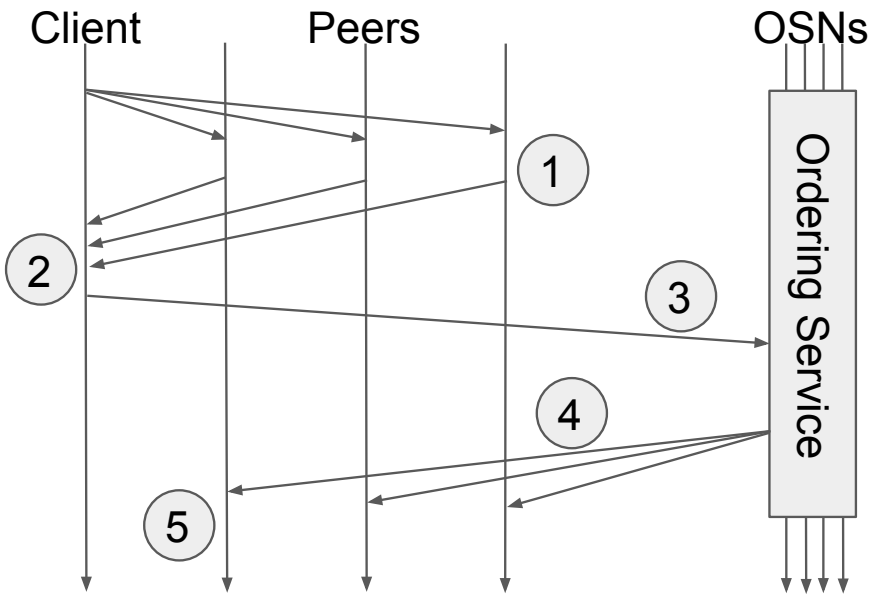
Hyperledger Fabric

- Permissioned blockchain platform [Androulaki et al., EuroSys'18]
- Modular design with **pluggable consensus**

HYPERLEDGER FABRIC

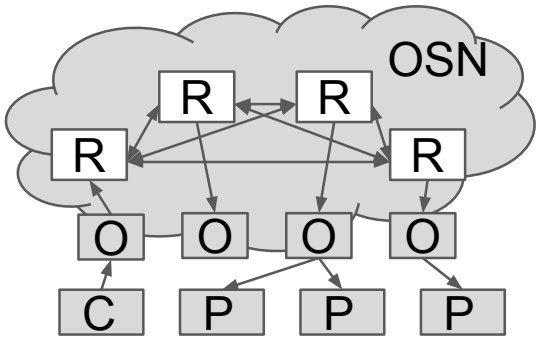
- **Transaction flow:**

1. Peers execute transaction
2. Client collects endorsements
3. Client sends transaction to be ordered
4. Ordering Service Nodes (OSNs) distribute blocks
5. Peers validate transactions



Ordering Services in Fabric

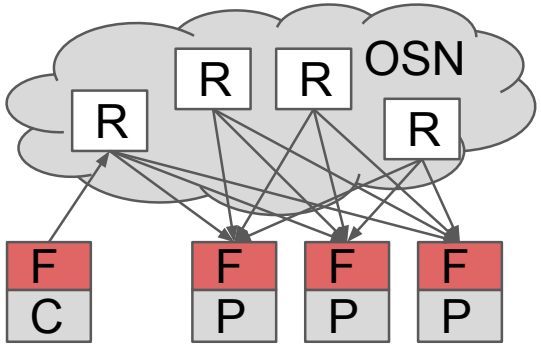
- No fault tolerance: Solo
- Crash-fault tolerance: Apache Kafka
- Byzantine fault tolerance: BFT-SMaRt
  [Sousa et al., DSN'18]



P: Peers C: Client R: Replica O: Orderer F: Frontend

Ordering Services in Fabric

- No fault tolerance: Solo
- Crash-fault tolerance: Apache Kafka
- Byzantine fault tolerance: BFT-SMaRt
  [Sousa et al., DSN'18]

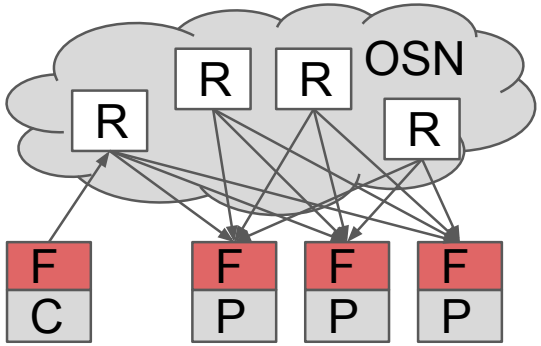


- BFT integration showed Fabric's shortcomings:
  - Client tasks compromise pluggability
  - **"Frontend"**: Fabric orderer now part of peer

P: Peers C: Client R: Replica O: Orderer F: Frontend

Ordering Services in Fabric

- No fault tolerance: Solo
- Crash-fault tolerance: Apache Kafka
- Byzantine fault tolerance: BFT-SMaRt
  [Sousa et al., DSN'18]



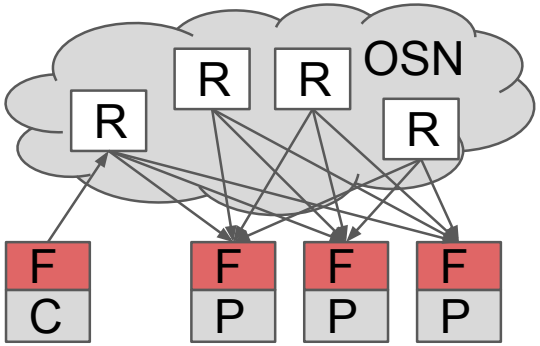
- BFT integration showed Fabric's shortcomings:
  - Client tasks compromise pluggability
  - **"Frontend"**: Fabric orderer now part of peer
  - Positioned on all peer nodes

  → Severely reduced modularity

P: Peers C: Client R: Replica O: Orderer F: Frontend

Ordering Services in Fabric

- No fault tolerance: Solo
- Crash-fault tolerance: Apache Kafka
- Byzantine fault tolerance: BFT-SMaRt
  [Sousa et al., DSN'18]



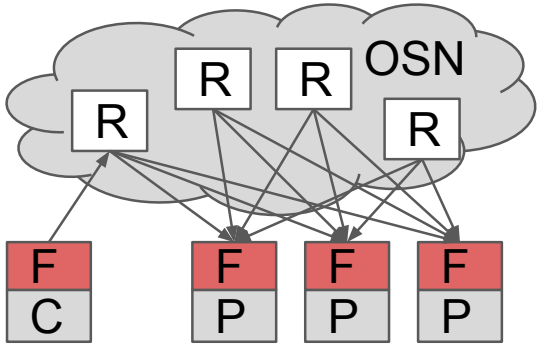
- BFT integration showed Fabric's shortcomings:
  - Client tasks compromise pluggability
  - **"Frontend"**: Fabric orderer now part of peer
  - Positioned on all peer nodes

  → Severely reduced modularity
  - Re-implementation for all protocols needed

P: Peers C: Client R: Replica O: Orderer F: Frontend

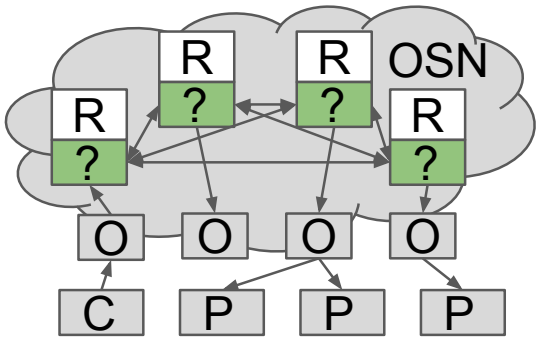
Ordering Services in Fabric

- No fault tolerance: Solo
- Crash-fault tolerance: Apache Kafka
- Byzantine fault tolerance: BFT-SMaRt
  [Sousa et al., DSN'18]



- BFT integration showed Fabric's shortcomings:
  - Client tasks compromise pluggability
  - **"Frontend"**: Fabric orderer now part of peer
  - Positioned on all peer nodes

  → Severely reduced modularity
  - Re-implementation for all protocols needed

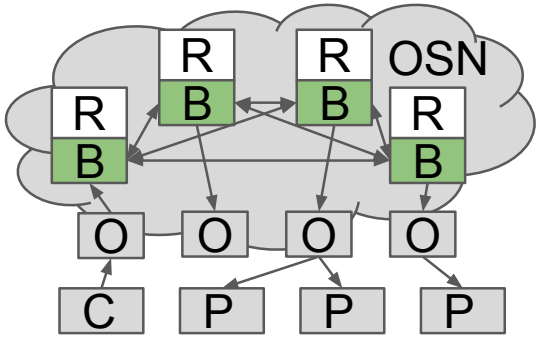


↳ **Goal**: Shift **BFT client** functionality to **replica nodes** to maintain modularity!

P: Peers C: Client R: Replica O: Orderer F: Frontend

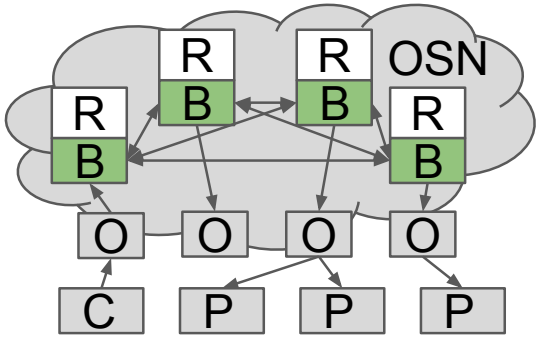
Bloxy: Blockchain-Aware Trusted Proxy

- **Consensus proxy** for Hyperledger Fabric
- Encapsulates BFT client functionality
- **Transparent** access to a **generic** ordering service



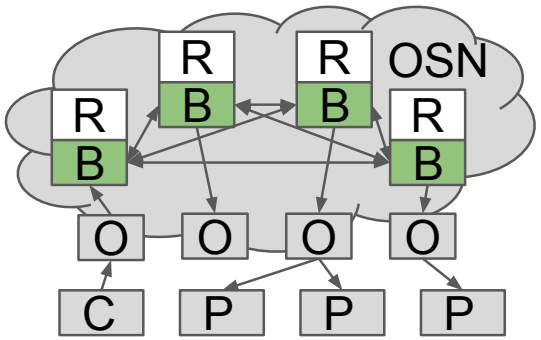
Bloxy: Blockchain-Aware Trusted Proxy

- **Consensus proxy** for Hyperledger Fabric
- Encapsulates BFT client functionality
- **Transparent** access to a **generic** ordering service
- Leverage **trusted execution environment** (TEE)
  - Hybrid fault model: TEE can only fail by crashing



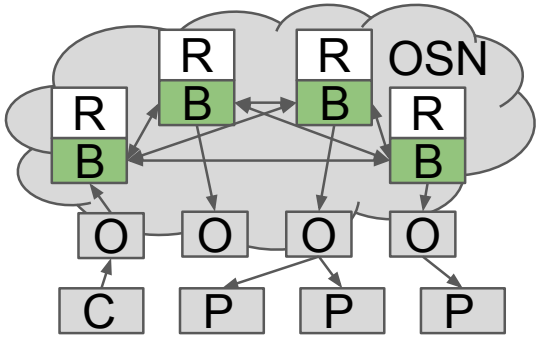
Bloxy: Blockchain-Aware Trusted Proxy

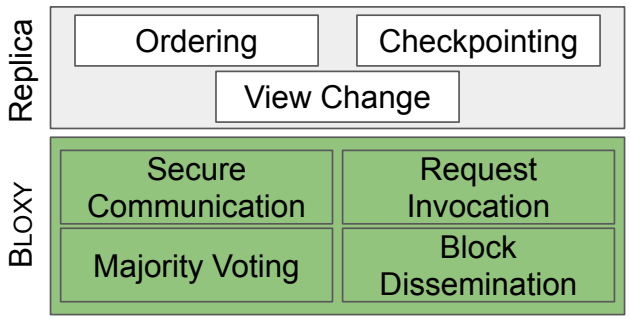
- **Consensus proxy** for Hyperledger Fabric
- Encapsulates BFT client functionality
- **Transparent** access to a **generic** ordering service
- Leverage **trusted execution environment** (TEE)
  - Hybrid fault model: TEE can only fail by crashing
- Integration of two BFT protocols:
  - **PBFT** [Castro et al., OSDI'99]
  - Hybrid protocol **Hybster** [Behl et al., EuroSys'17]



BLOXY: Blockchain-Aware Trusted Proxy

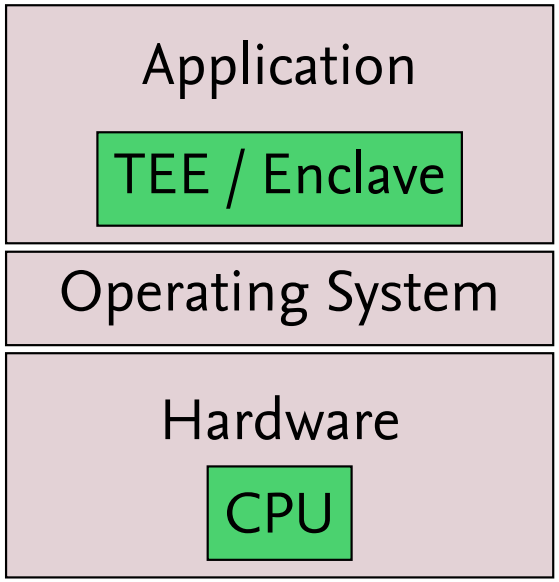
- **Consensus proxy** for Hyperledger Fabric
- Encapsulates BFT client functionality
- **Transparent** access to a **generic** ordering service
- Leverage **trusted execution environment** (TEE)
  - Hybrid fault model: TEE can only fail by crashing
- Integration of two BFT protocols:
  - **PBFT** [Castro et al., OSDI'99]
  - Hybrid protocol **Hybster** [Behl et al., EuroSys'17]
- Functionality:
  1. Establishing secure connections
  2. Forwarding transactions as BFT requests
  3. Majority voting on replies
  4. Block dissemination





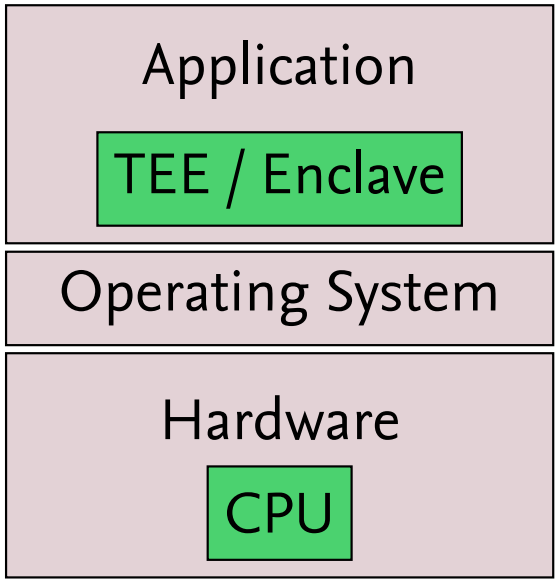
Trusted Execution: Intel SGX in a Nutshell

- x86 **instruction set extension** introduced with Skylake architecture
- Creation of TEEs $\rightarrow$ **enclaves**
- Execution and data inside enclaves **protected from privileged software**
- Hardware-based **memory integrity protection** and **encryption**
- Remote attestation of enclaves
- **Only CPU** is trusted



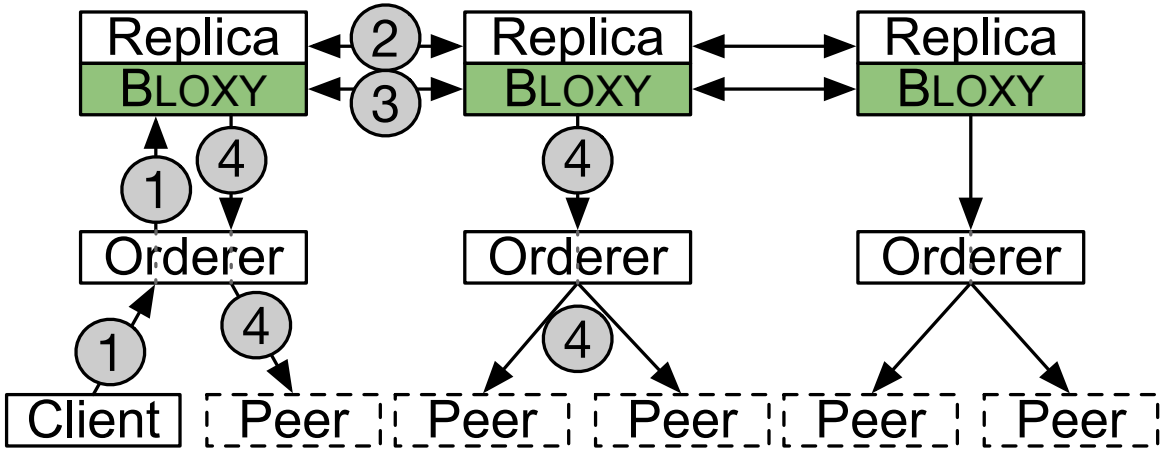
Trusted Execution: Intel SGX in a Nutshell

- x86 **instruction set extension** introduced with Skylake architecture
- Creation of TEEs $\rightarrow$ **enclaves**
- Execution and data inside enclaves **protected from privileged software**
- Hardware-based **memory integrity protection** and **encryption**
- Remote attestation of enclaves
- **Only CPU** is trusted



↳ **Intel SGX** allows the creation of **enclaves**, trusted execution environments (TEEs) protected by hardware.

Bloxy Transaction Flow



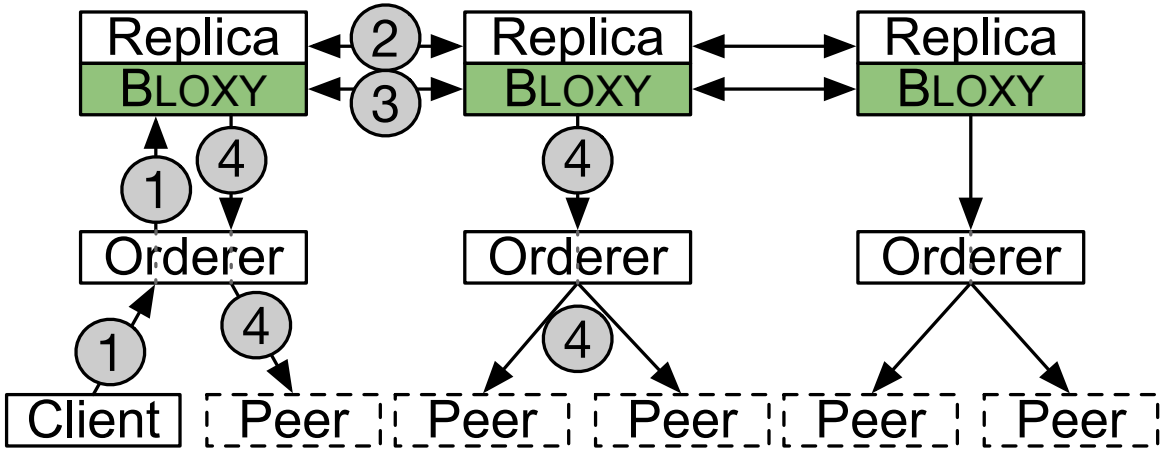
① Transaction forwarded to Bloxy via orderer

② Replicas order translated BFT request from Bloxy

③ Bloxies get block, exchange hashes, perform majority voting

④ Blocks are disseminated to all connected nodes

Bloxy Transaction Flow



① Transaction forwarded to Bloxy via orderer

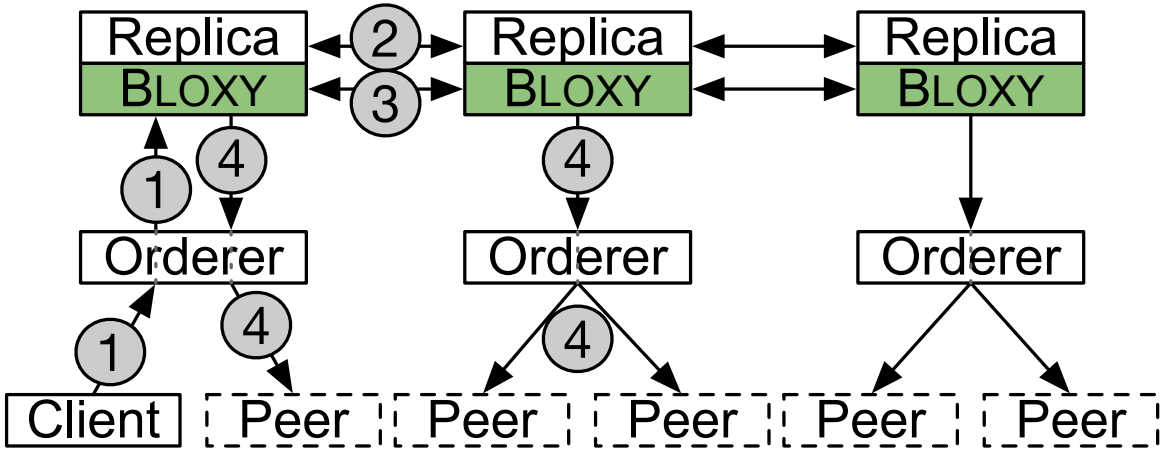
② Replicas order translated BFT request from Bloxy

③ Bloxies get block, exchange hashes, perform majority voting

④ Blocks are disseminated to all connected nodes

→ Transmit only hashes and one block per orderer

BLOXY Transaction Flow



① Transaction forwarded to BLOXY via orderer

② Replicas order translated BFT request from BLOXY

③ BLOXIES get block, exchange hashes, perform majority voting

④ Blocks are disseminated to all connected nodes

→ Transmit only hashes and one block per orderer

↳ Lower overhead due to **less** and **shorter** messages!

Bloxy Short-Term Block Storage

- Fulfil Fabric's **requirements** for ordering services
- **No Skipping**: **all** peers need **all** blocks
  - Byzantine orderers cannot manipulate blocks signed by Bloxy
  - How to ensure Byzantine orderers **deliver** all blocks?

Bloxy Short-Term Block Storage

- Fulfil Fabric's **requirements** for ordering services
- **No Skipping**: **all** peers need **all** blocks
  - Byzantine orderers cannot manipulate blocks signed by Bloxy
  - How to ensure Byzantine orderers **deliver** all blocks?
- Bloxy needs to ensure at least one correct orderer receives each block
- Block stored on replicas until $f + 1$ orderers acknowledge reception

Bloxy Short-Term Block Storage

- Fulfil Fabric's **requirements** for ordering services
- **No Skipping: all** peers need **all** blocks
  - Byzantine orderers cannot manipulate blocks signed by Bloxy
  - How to ensure Byzantine orderers **deliver** all blocks?
- Bloxy needs to ensure at least one correct orderer receives each block
- Block stored on replicas until $f + 1$ orderers acknowledge reception
- Orderer can query **Short-Term Block Storage:**
  - Receive missing block $\rightarrow$ full round of BFT communication
  - Receive list of acknowledged orderers to query

Bloxy Short-Term Block Storage

- Fulfil Fabric's **requirements** for ordering services
- **No Skipping: all** peers need **all** blocks
  - Byzantine orderers cannot manipulate blocks signed by Bloxy
  - How to ensure Byzantine orderers **deliver** all blocks?
- Bloxy needs to ensure at least one correct orderer receives each block
- Block stored on replicas until $f + 1$ orderers acknowledge reception
- Orderer can query **Short-Term Block Storage:**
  - Receive missing block $\rightarrow$ full round of BFT communication
  - Receive list of acknowledged orderers to query
- Performance vs. fault tolerance $\rightarrow$ orderer's behaviour is configurable

Bloxy Short-Term Block Storage

- Fulfil Fabric's **requirements** for ordering services
- **No Skipping: all** peers need **all** blocks
  - Byzantine orderers cannot manipulate blocks signed by Bloxy
  - How to ensure Byzantine orderers **deliver** all blocks?
- Bloxy needs to ensure at least one correct orderer receives each block
- Block stored on replicas until $f + 1$ orderers acknowledge reception
- Orderer can query **Short-Term Block Storage:**
  - Receive missing block $\rightarrow$ full round of BFT communication
  - Receive list of acknowledged orderers to query
- Performance vs. fault tolerance $\rightarrow$ orderer's behaviour is configurable

↳ Bloxy can guarantee **all requirements** for ordering services!

Generic Consensus

- **Generality:** BLOXY is not limited to one specific protocol
- BLOXY implemented as a **library**
- BFT frameworks can include the BLOXY library

Generic Consensus

- **Generality:** BLOXY is not limited to one specific protocol
- BLOXY implemented as a **library**
- BFT frameworks can include the BLOXY library
- Call BLOXY for **trusted operations** shared by most BFT frameworks:
  - Connection establishment
  - Request translation
  - Majority voting
- **No functional changes** to BFT framework necessary
  - Core aspects are unchanged: ordering, checkpointing, view change

Generic Consensus

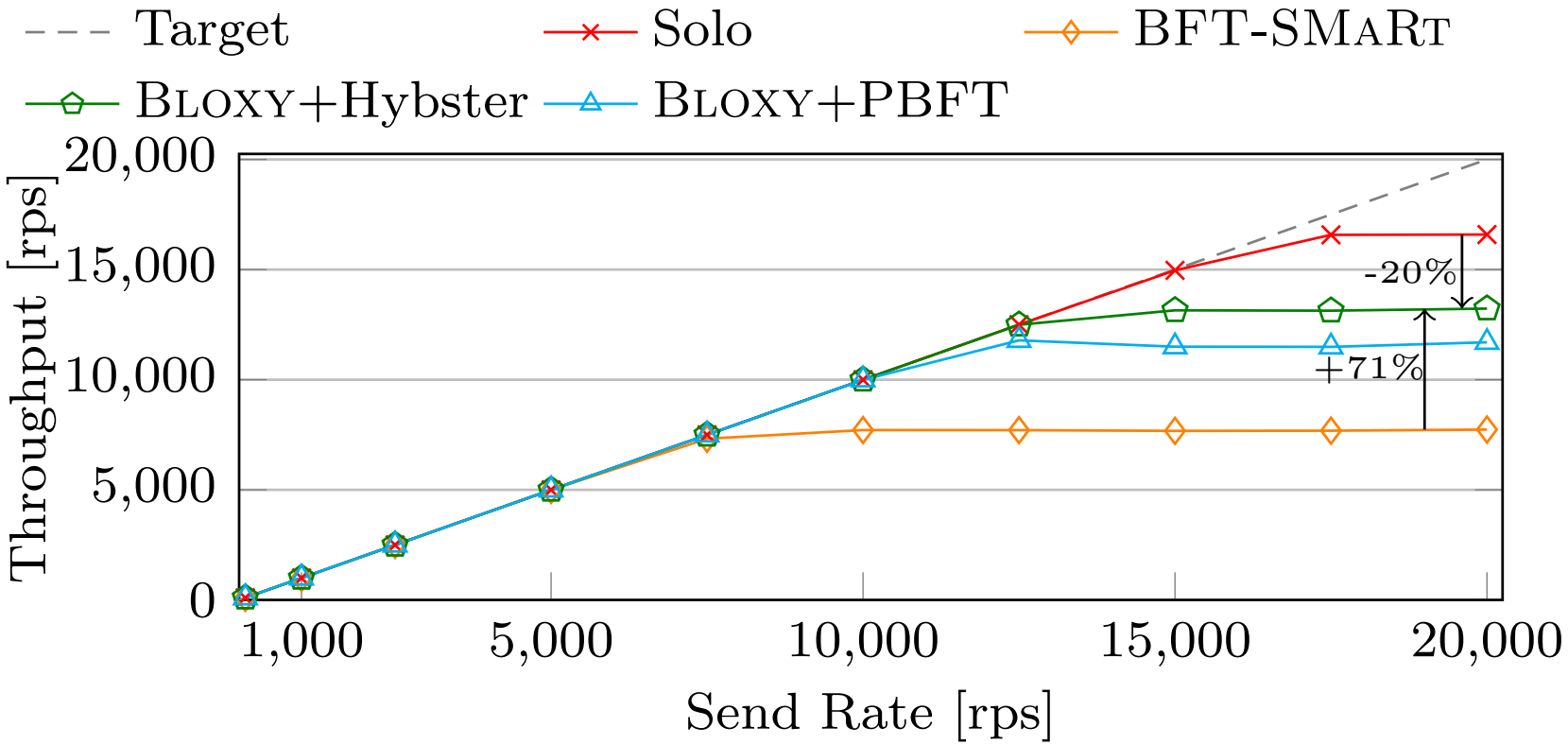
- **Generality:** BLOXY is not limited to one specific protocol
- BLOXY implemented as a **library**
- BFT frameworks can include the BLOXY library
- Call BLOXY for **trusted operations** shared by most BFT frameworks:
  - Connection establishment
  - Request translation
  - Majority voting
- **No functional changes** to BFT framework necessary
  - Core aspects are unchanged: ordering, checkpointing, view change

↳ BLOXY allows easy integration of **arbitrary BFT protocols**!

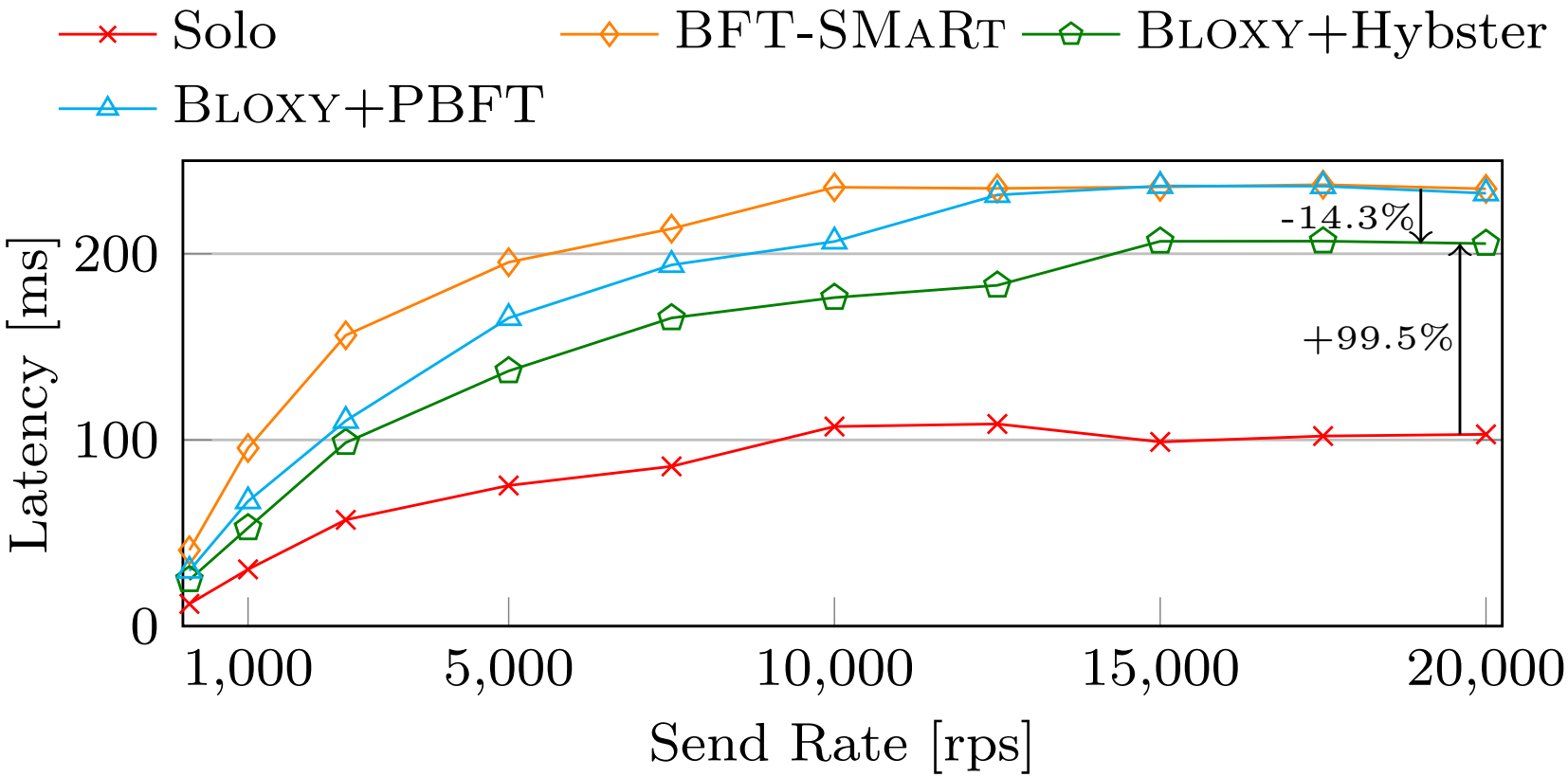
Evaluation of BLOXY

- 4 machines for **ordering service**
  - SGX-capable 4-core Xeon v5 CPUs, 32GB RAM
- 5 machines as **peer and orderer** nodes
  - non-SGX 4-core Xeon v2 CPUs, 16GB RAM
- **10 Gbps** interconnection (switched network)
- **Ordering Service Component:**
  - What is BLOXY's **impact on latency**?
  - What **throughput** can BLOXY achieve?
- **Complete Fabric network:**
  - Does BLOXY affect the performance of **Fabric**?

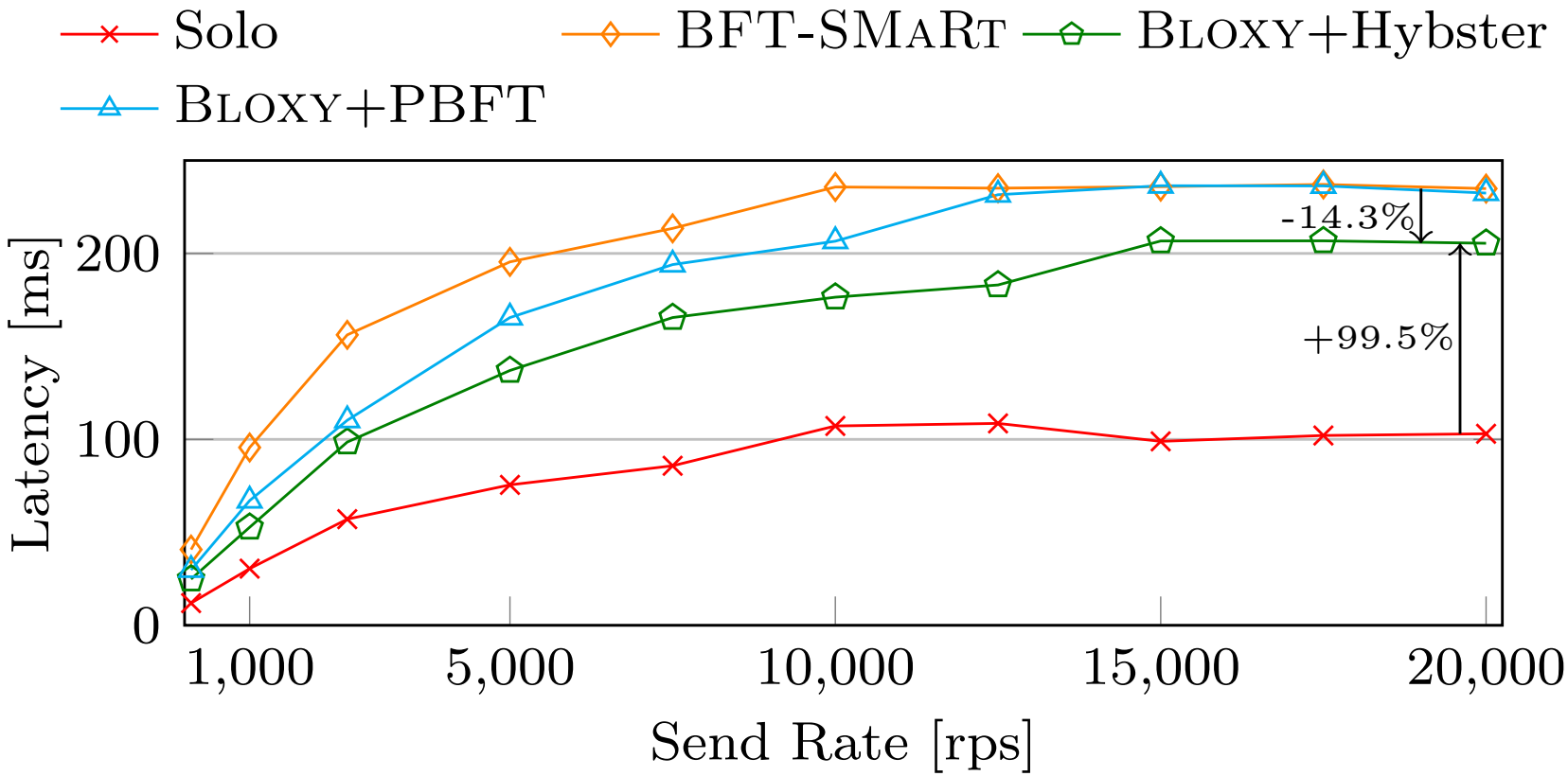
Throughput Evaluation of Ordering Services



Latency Evaluation of Ordering Services

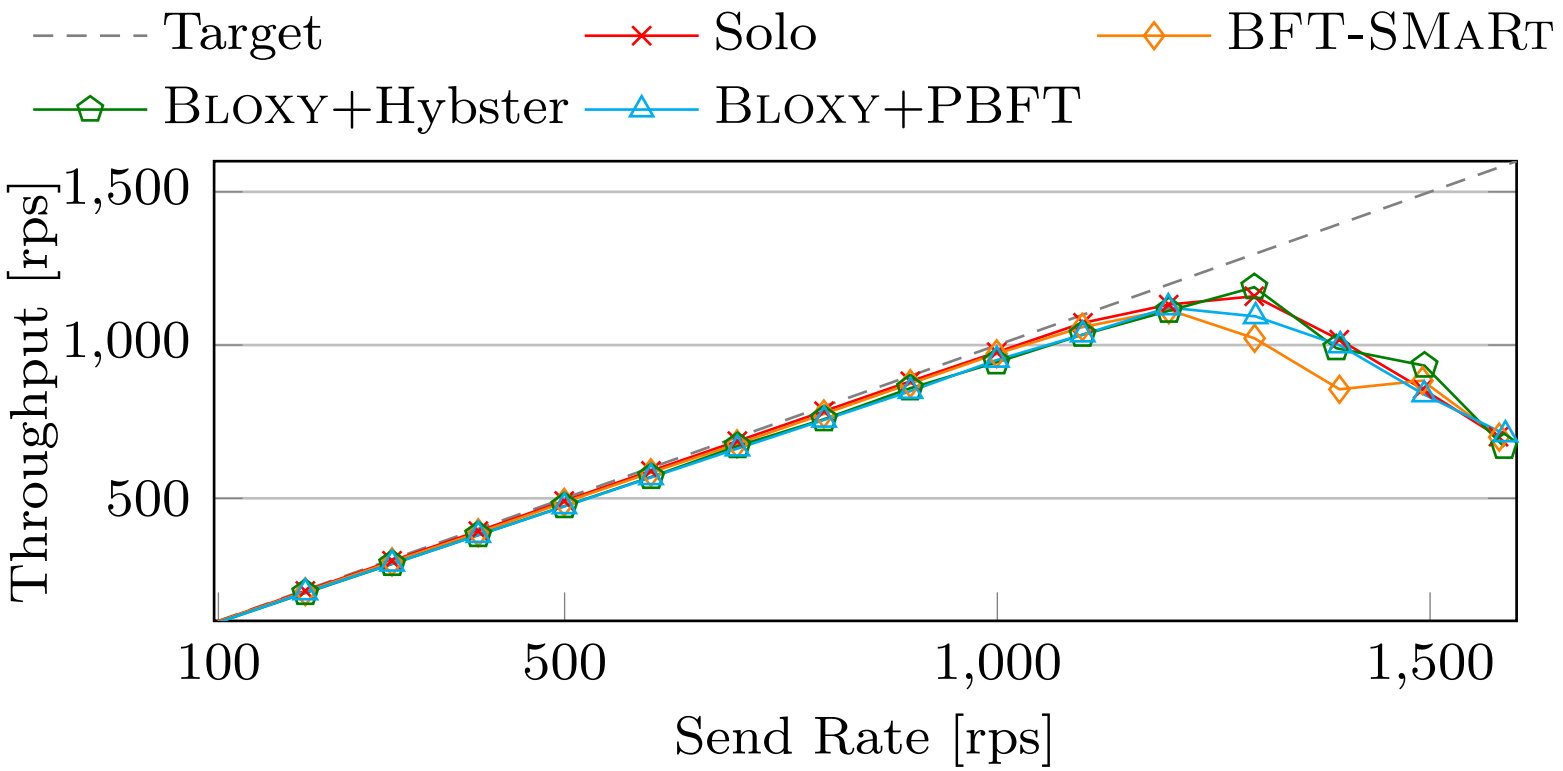


Latency Evaluation of Ordering Services



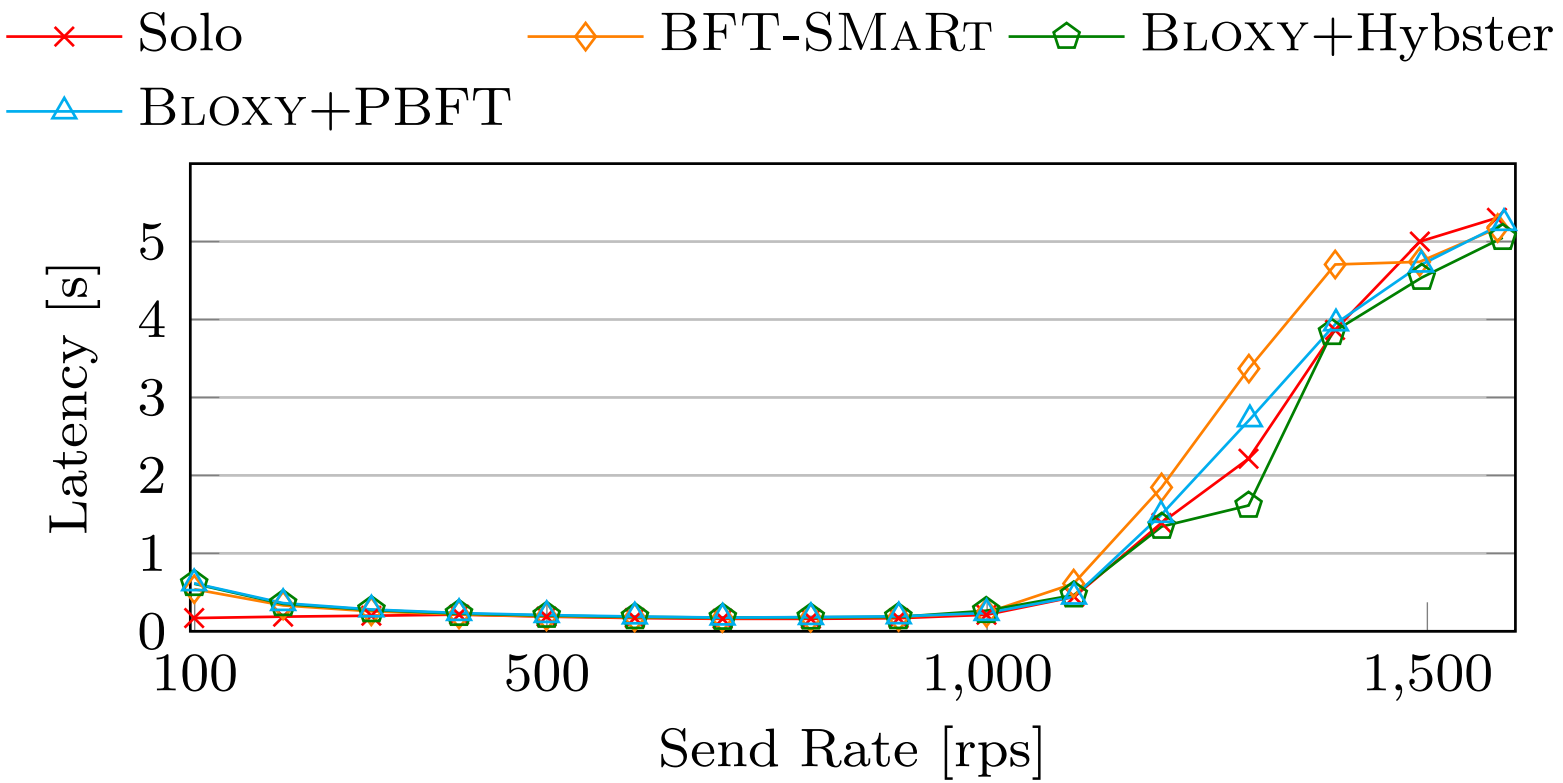
↳ Lower communication overhead of Bloxy protocols, **improved** throughput and latency

Throughput Evaluation of Fabric Network

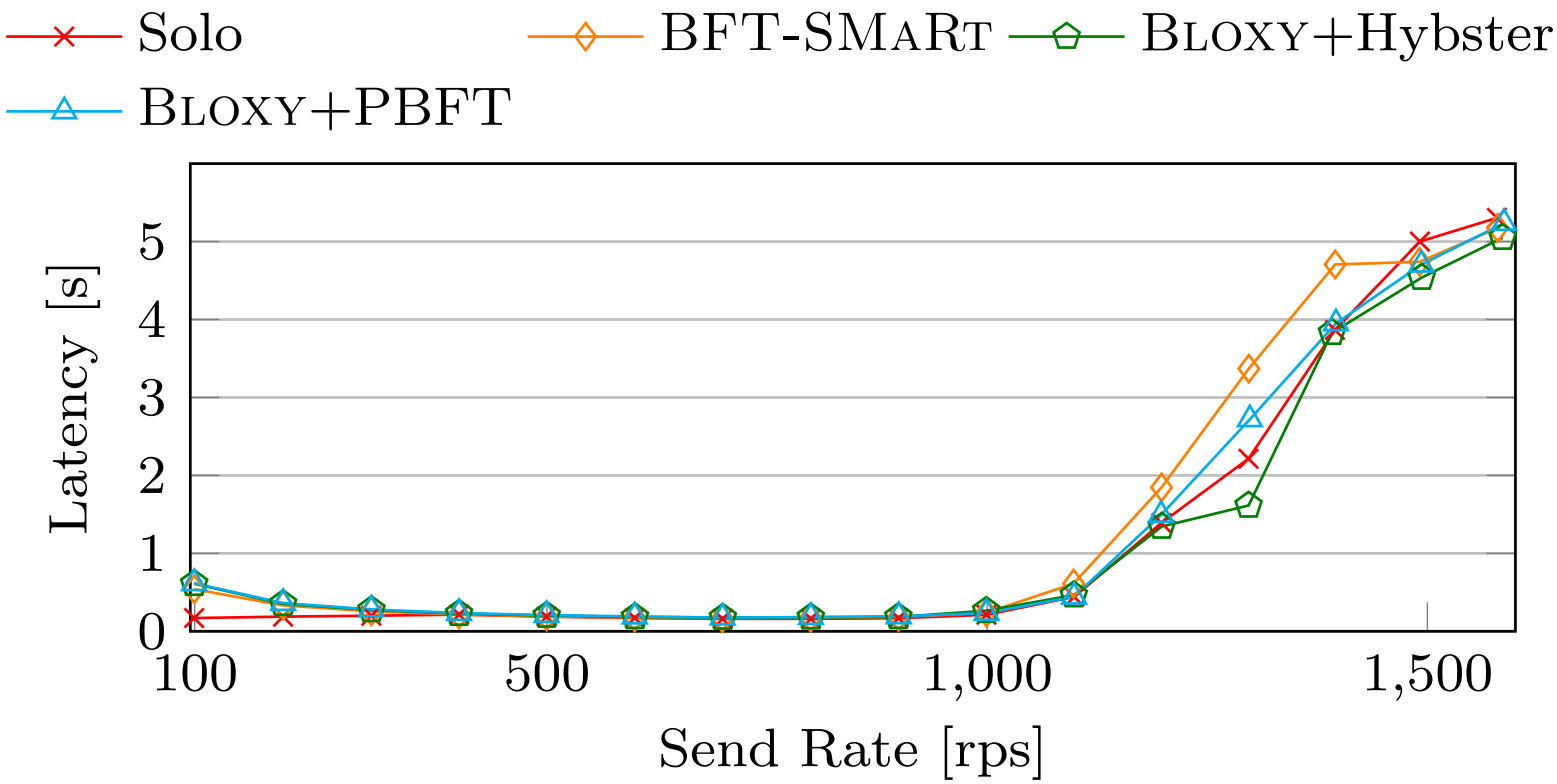


- Hyperledger Caliper with full Fabric network
  - 12 clients, 2 peers, 3-4 replicas in ordering service
  - Bloxy-backed protocols give comparable performance

Latency Evaluation of Fabric Network



Latency Evaluation of Fabric Network



↳ Ordering service has **no noticeable impact** on Fabric's performance

↳ Generic consensus at **no performance cost**!

→ Full evaluation of all ordering services in paper!

Related Work

- Other uses of Intel SGX as trusted proxy
- **Troxy** [Li et al., DSN'18]
  - Shift BFT client functionality to replica side
  - Aid adoption of BFT protocols e. g. for SMTP and HTTP
  - No changes to common protocols and client implementations necessary
- LibSeal [Aublin et al., EuroSys'18]
  - Audit library for detecting integrity violations
- VeritasDB [Sinha et al., IACR Cryptology ePrint'18]
  - Network proxy for key-value stores with integrity guarantees

Related Work

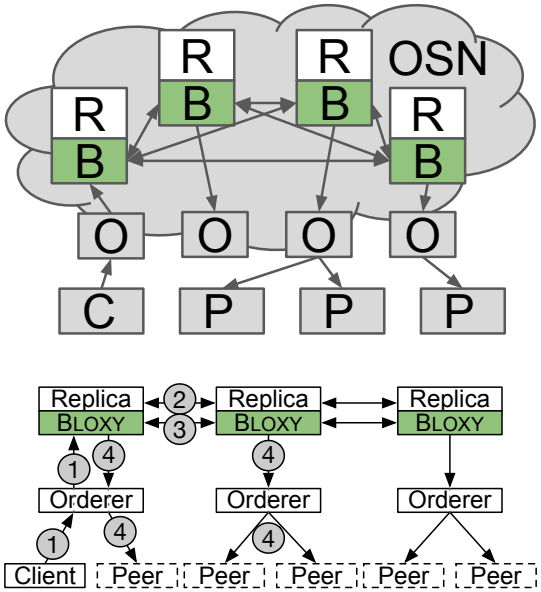
- Other uses of Intel SGX as trusted proxy
- **Troxy** [Li et al., DSN'18]
  - Shift BFT client functionality to replica side
  - Aid adoption of BFT protocols e. g. for SMTP and HTTP
  - No changes to common protocols and client implementations necessary
- LibSeal [Aublin et al., EuroSys'18]
  - Audit library for detecting integrity violations
- VeritasDB [Sinha et al., IACR Cryptology ePrint'18]
  - Network proxy for key-value stores with integrity guarantees

↳ BLOXY tailored for **blockchain** use case and communication, integration into a permissioned blockchain with ordering service

Conclusion

BLOXY's contributions:

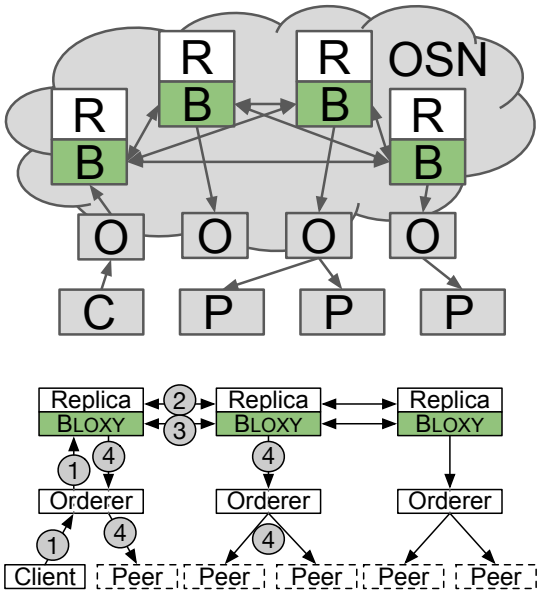
- **Consensus proxy** for Hyperledger Fabric
- Encapsulation of **BFT client functionality**



Conclusion

BLOXY's contributions:

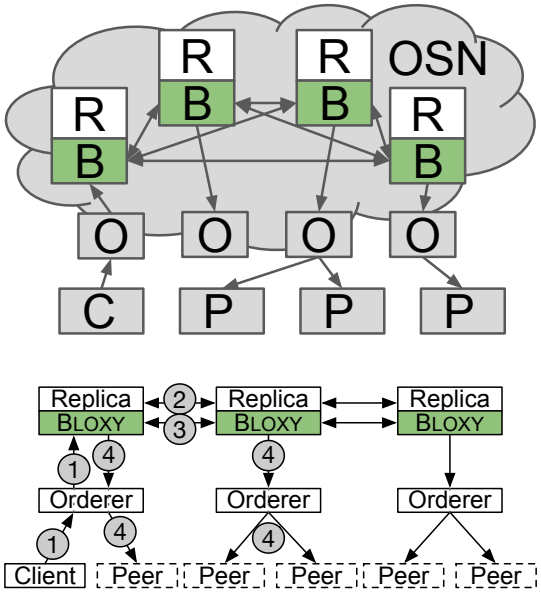
- **Consensus proxy** for Hyperledger Fabric
- Encapsulation of **BFT client functionality**
- Enabling **easy integration** of ordering services
- Allowing operators to choose **suitable** protocols



Conclusion

BLOXY's contributions:

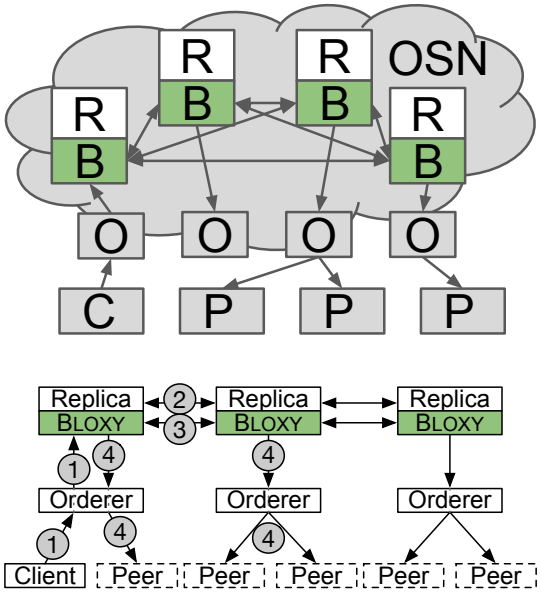
- **Consensus proxy** for Hyperledger Fabric
- Encapsulation of **BFT client functionality**
- Enabling **easy integration** of ordering services
- Allowing operators to choose **suitable** protocols
- BLOXY maintains Fabric's **modularity**
- **Good performance** compared to directly integrated BFT protocols



Conclusion

BLOXY's contributions:

- **Consensus proxy** for Hyperledger Fabric
- Encapsulation of **BFT client functionality**
- Enabling **easy integration** of ordering services
- Allowing operators to choose **suitable** protocols
- BLOXY maintains Fabric's **modularity**
- **Good performance** compared to directly integrated BFT protocols



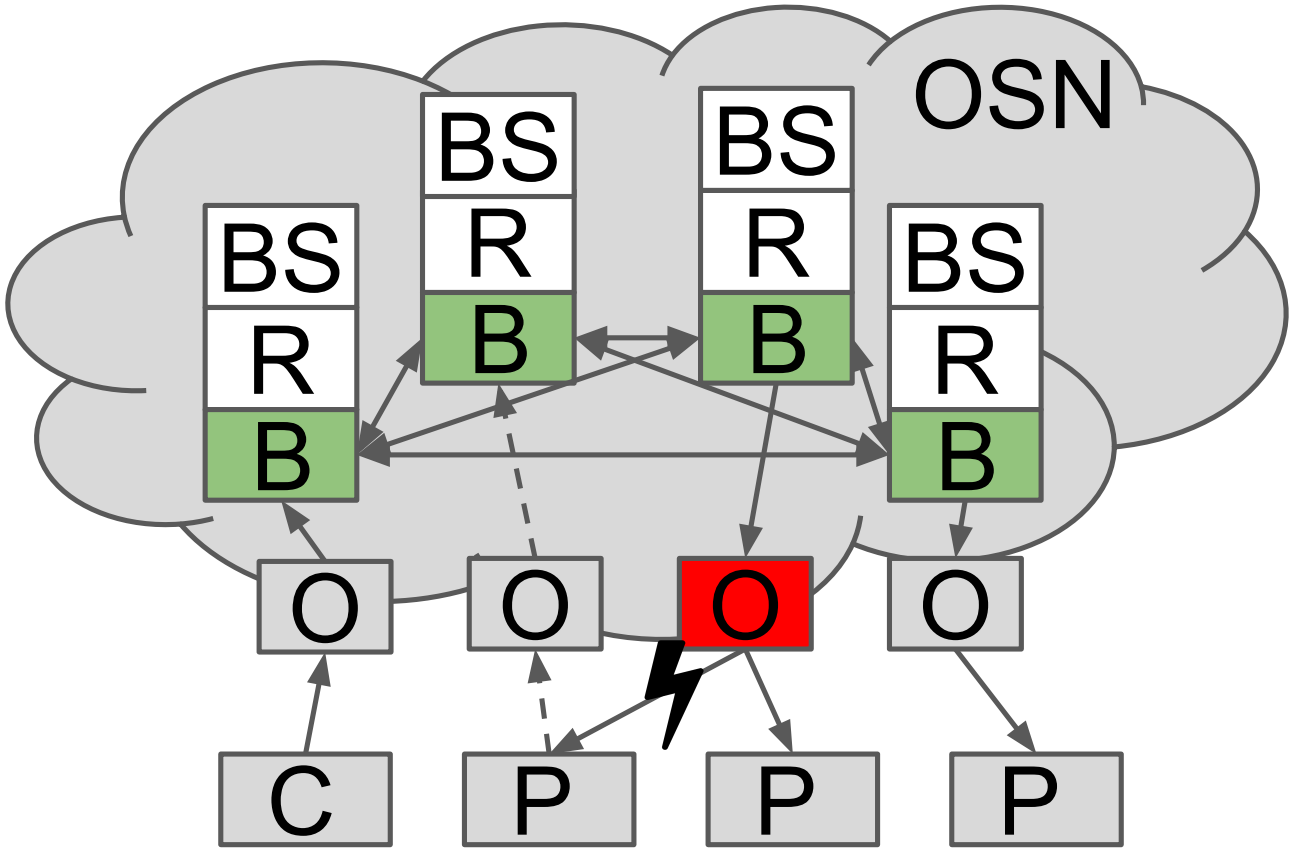
Thank you for your attention! Questions?

ruesch@ibr.cs.tu-bs.de

Requirements of Fabric's Ordering Services

- Agreement:
  - Blocks with same sequence number should be identical
  - No support of forks
- Hashchain integrity:
  - Blocks contain hash of predecessor
  - Blocks form non-manipulatable hashchain
- No skipping:
  - No peer can skip a block
  - Has to receive all previous blocks
- No creation of transactions
- Liveness:
  - Transactions submitted to the ordering service will eventually be contained in block

Bloxy Short-Term Block Storage



P: Peers C: Client R: Replica O: Orderer B: Bloxy BS: Block Storage

Bloxy Short-Term Block Storage: Evaluation Results

	Bloxy		(iii) Orderer
	(i) Storage Hit	(ii) Storage Miss	
Duration	21.79 ms	7.19 ms	7.28 ms

(i) Query Short-Term Block Storage

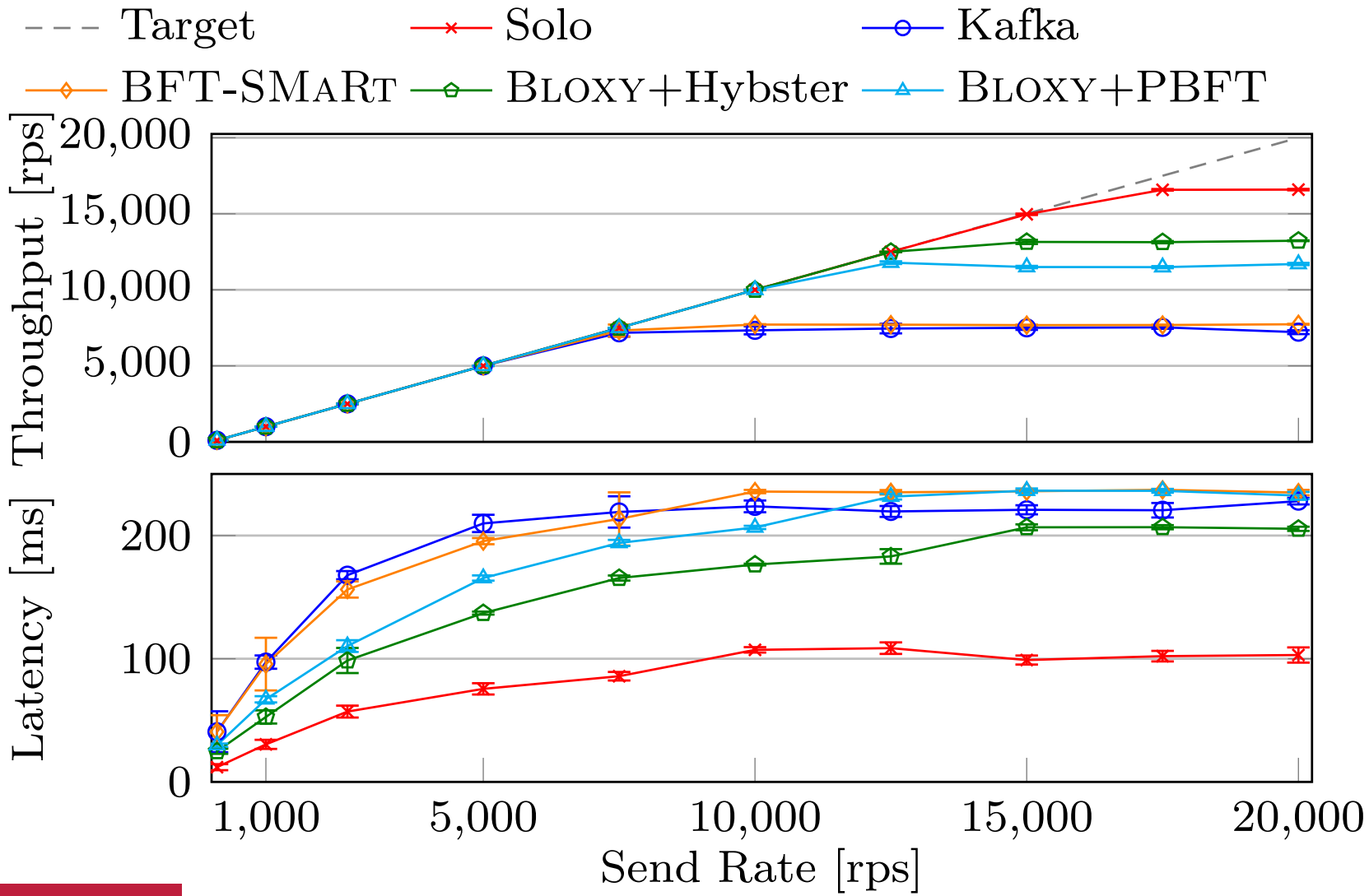
(ii) Query Block Storage, query orderer from acknowledged list

(iii) Randomly query orderers

Public API of Bloxy

- Connection establishment
- Orderer:
  - Submit transaction for ordering
  - Receive voted block
- Replicas:
  - Receive forwarded and translated transaction for ordering
  - Submit block for majority voting

Ordering Service



Ordering Service

- Throughput:
  - 71 % (Hybster) and 51 % (PBFT) higher compared to BFT-SMaRt
  - 20 % (Hybster) and 29.5 % (PBFT) lower compared to Solo
- Latency:
  - 99 % (Hybster) and 125 % (PBFT) higher than Solo
  - 14.3 % (Hybster) lower than BFT-SMaRt

Full Fabric Network