



MONASH
University



Analyzing the Performance of the Inter-Blockchain Communication Protocol

[João Otávio Chervinski* †](#), [Jiangshan Yu*](#), [Diego Kreutz‡](#), [Sherry Xu‡](#)

*Monash University, Australia

† CSIRO Data61, Australia

‡ Federal University of Pampa, Brazil

Blockchains and performance


- **What forecasts say for blockchains:**
 - Will grow at a rate of 85.9% from 2022 to 2030 (*Triple A and Grandview Research*)
 - Will reach \$67.4 billion global market by 2026 (*Markets and Markets*)
 - Are forecasted to generate over \$3.1 trillion in business value by 2030 (Gartner)
- **Despite the large number of applications that are adopting blockchains, some long-standing issues still remain:**
 - Low throughput
 - High transaction confirmation latency
 - Trade-offs (security, scalability, decentralization)
- **Early blockchains are known to have those problems, however, recently developed blockchain systems also suffer from those issues, despite existing work.**



Improving blockchain performance

- In “*BLOCKBENCH: A Framework for Analyzing Private Blockchains*”, Dinh et al. show that:
 - Hyperledger Fabric v0.6 is **unable to scale beyond 16 nodes** due to implementation issues in its consensus protocol.
- In “*Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform*”, Thakkar et al. show that:
 - Cryptography operations and REST API calls led to transaction validations bottlenecks in Hyperledger Fabric v1.0. **Proposed optimizations improve throughput by 16x.**
- In “*Diablo: A Benchmark Suite for Blockchains*”, Gramoli et. al observe:
 - There’s a **large difference between claimed blockchain performance and performance in real world scenarios** (Algorand 50x lower, Solana 22x lower, Avalanche 13x lower)

Blockchain performance

- The performance of isolated blockchains has been extensively studied. Being researchers, we like to study new stuff!
- Given the increase in number of blockchains and services, the **ability to transfer information between them** became desirable.
- Many proposals for cross-chain communication such as relays, sidechains and atomic swap protocols.  Same problem as before
- Now we have generic protocols that aim to transfer arbitrary data (XCMP and IBC). Unlike operations in isolated blockchains, the performance of **cross-chain communication** has not been given much attention.



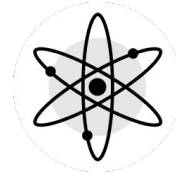
Cross-chain communication protocols



XCMP Protocol

Still under development

Replaced by a **resource demanding temporary protocol** for the time being



C Ø S M O S

IBC Protocol

Used to connect 53 blockchains

\$30.3 billion volume in 2022

Bridging Cosmos and Polkadot

Supports Interchain accounts (recent)

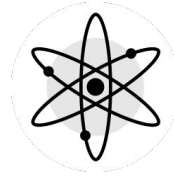
Cross-chain communication protocols



XCM Protocol

Still under development

Replaced by a **resource demanding temporary protocol** for the time being



C Ø S M O S

IBC Protocol

Used to connect 53 blockchains

\$30.3 billion volume in 2022

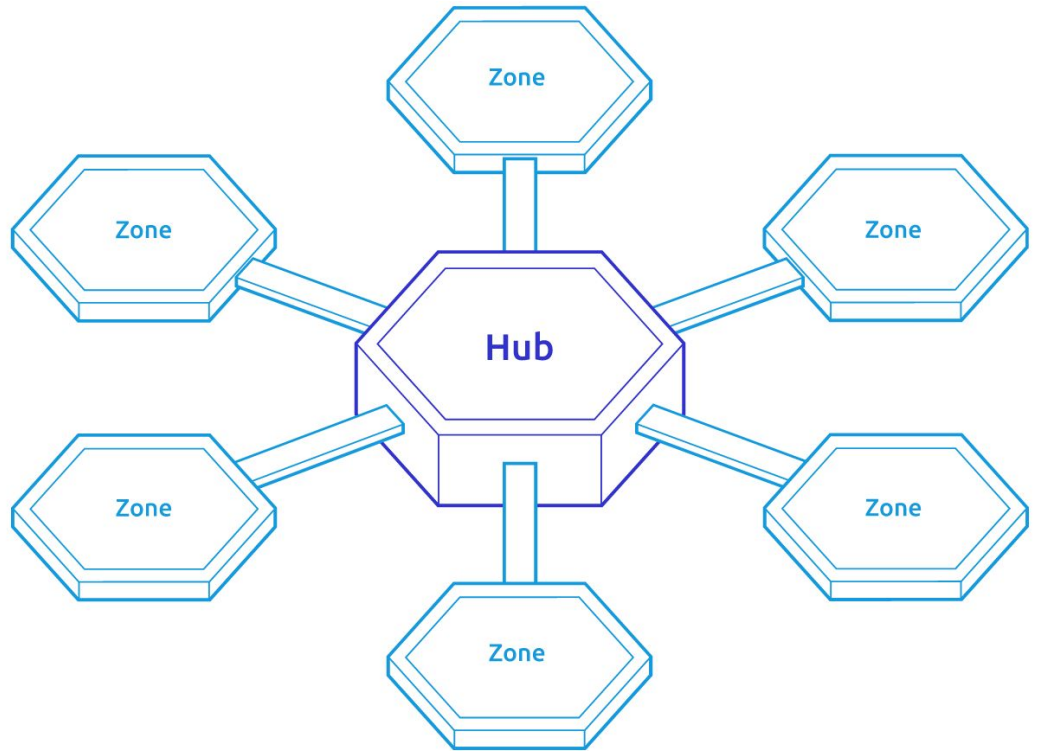
Bridging Cosmos and Polkadot

Supports Interchain accounts (recent)

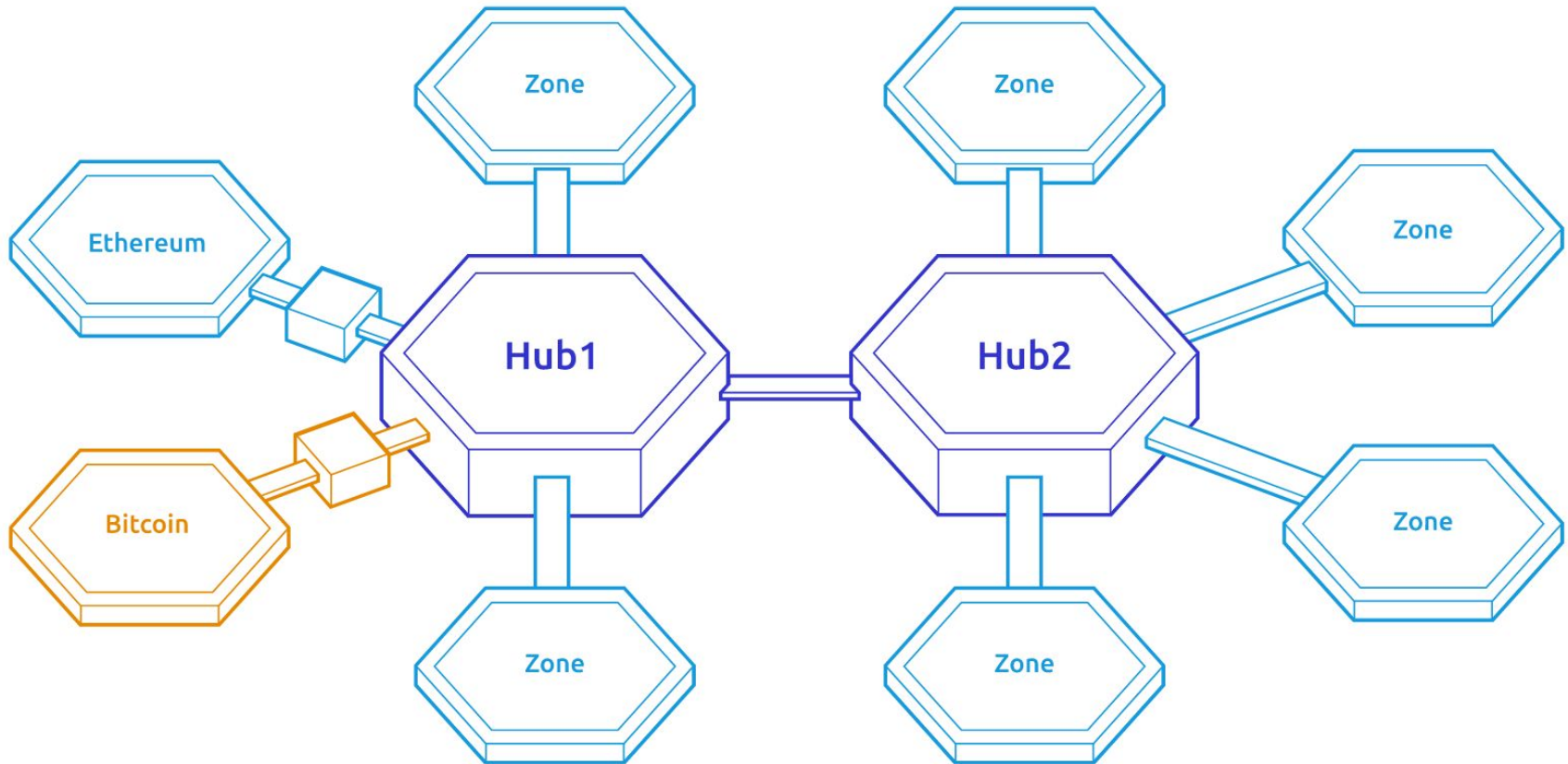


Cosmos (in a nutshell)

- Blockchains are called **zones**
- **Hubs** are special types of zones serve as a point of connection
- Any zone can also be a **Hub**
- **Hubs** reduce the need for connections

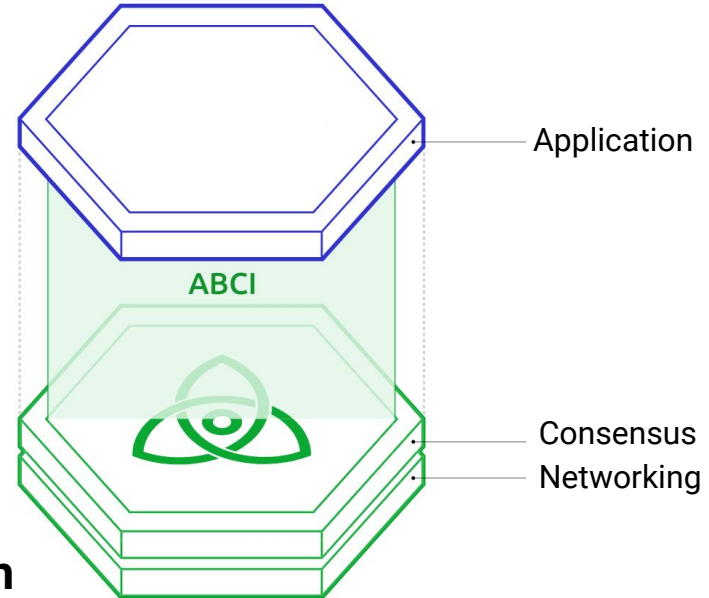


Cosmos (in a nutshell)

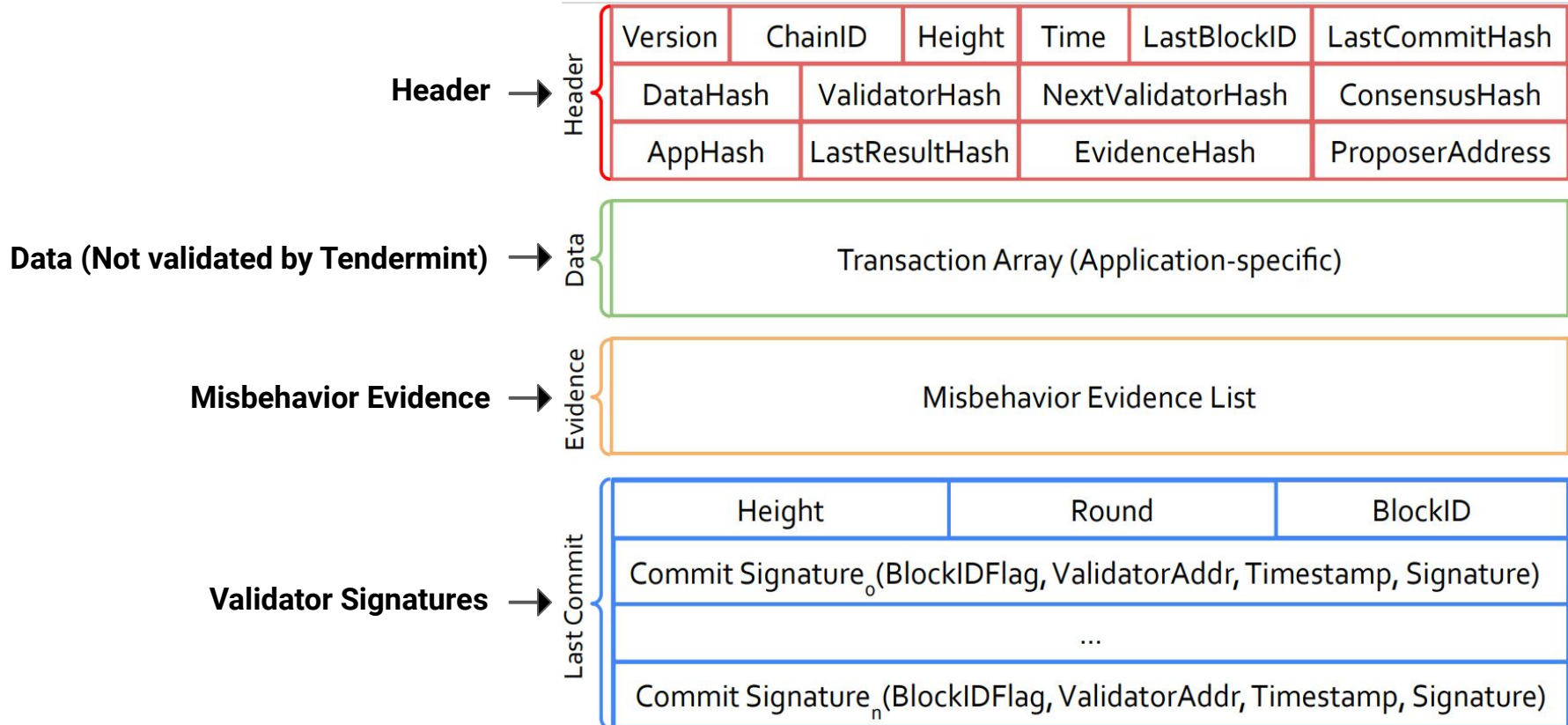


Tendermint

- Cosmos blockchains are built using **Tendermint** and the **Cosmos SDK**
- Tendermint is composed by:
 - **Tendermint Core**
 - Tendermint BFT (consensus)
 - P2P networking protocol
 - **Application BlockChain Interface (ABCI)**
 - A generic interface that allows applications to communicate with Tendermint
- **The Cosmos SDK provides the application with modules that implement *authentication, staking, slashing, transfers, IBC* and others.**



Tendermint block structure



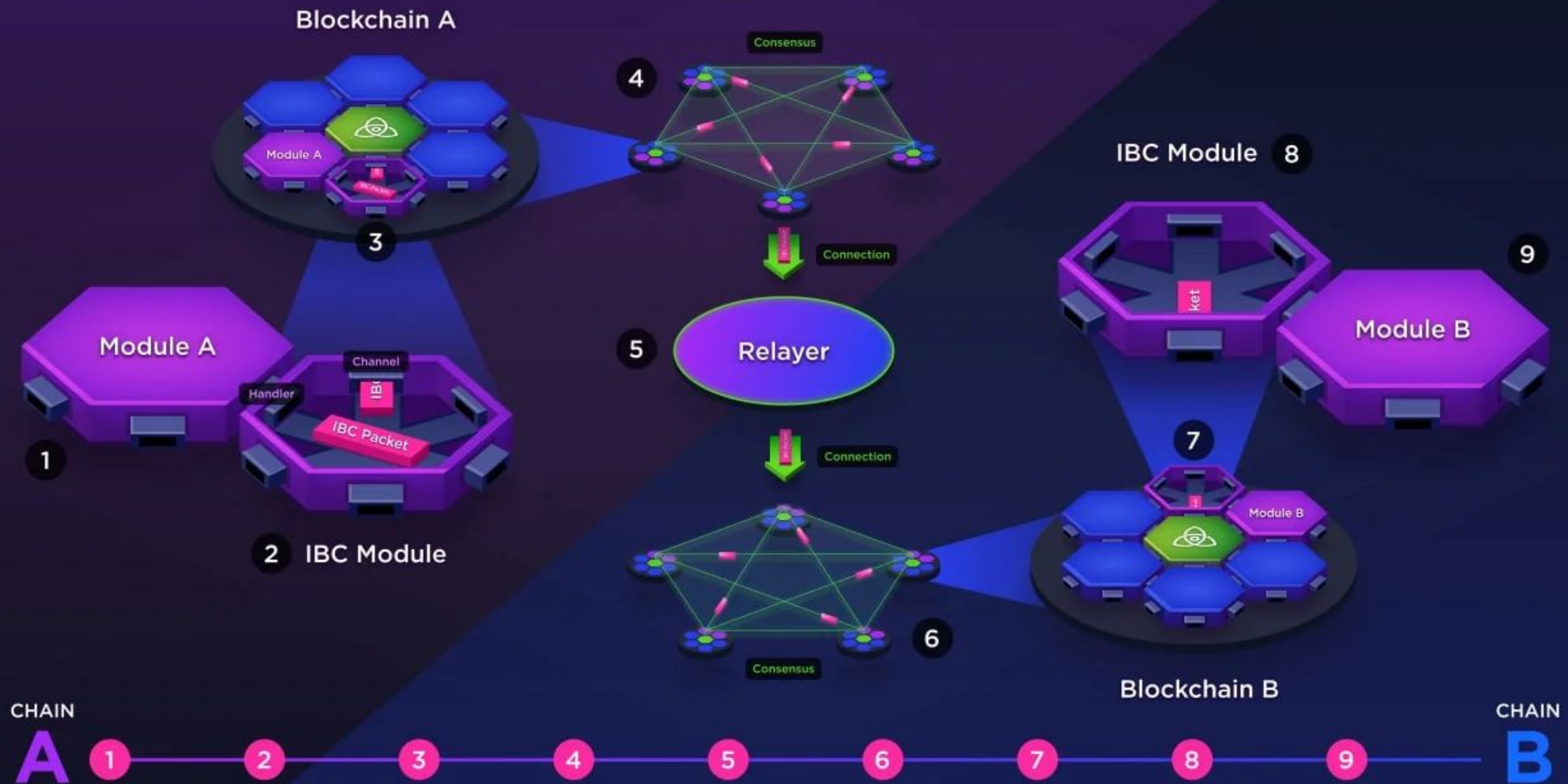
The Inter-Blockchain Communication (IBC) Protocol

- End-to-end, connection oriented, stateful communication protocol
- IBC handles **authentication, transport and ordering** of opaque data packets between IBC modules in separate ledgers
- Communication through IBC requires a channel to be established between two communicating blockchains:
 - They require a connection established through a handshaking process
 - They work as routes for message delivery between two ledgers
 - Channels are maintained by one or more external applications called **relayers**
- Relayers deliver data across two ledgers by scanning the ledgers, constructing datagrams and submitting them to the opposite ledger
 - Relaying is **permissionless**, all verification is performed by the ledgers

IBC is the Inter-Blockchain
Communication Protocol

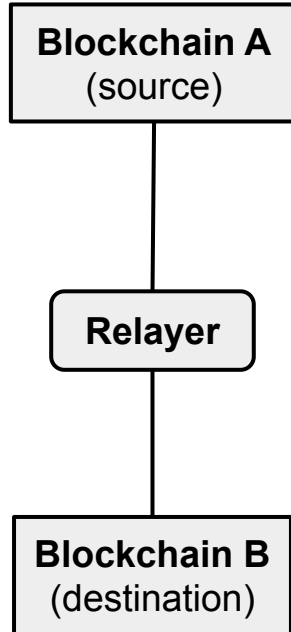
IBC

A visualization of an IBC packet
traveling between two blockchains



Fungible token transfer using IBC

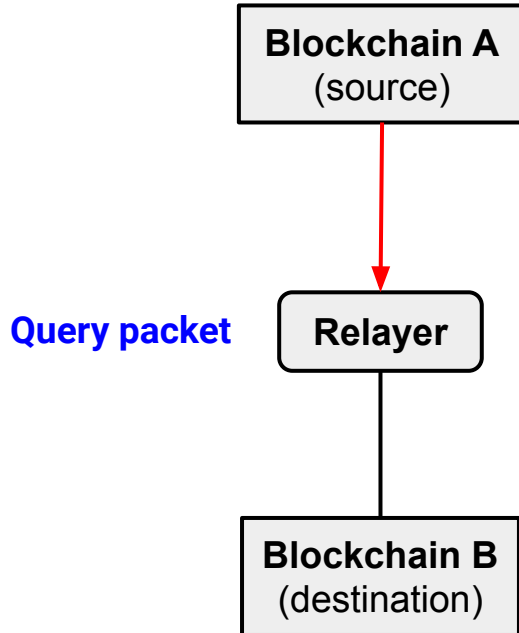
Commit TransferMsg



Token transfers are composed by 3 steps:

- **TransferMsg:** Requests a fungible token transfer. Stores a commitment to the packet data and timeout in the source chain.
- **MsgRecvPacket:** Informs that a packet has been received and processed after verification of its commitment proof in the source chain. Stores a proof of packet acknowledgement in the destination chain.
- **MsgAcknowledgement:** Informs that a packet has been processed and acknowledged in the destination chain.

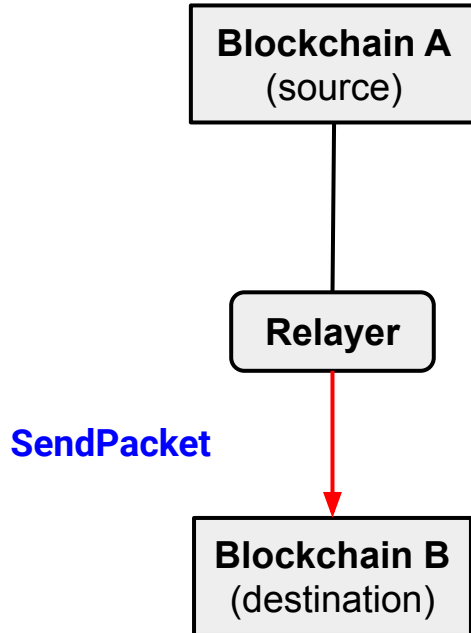
Fungible token transfer using IBC



Token transfers are composed by 3 steps:

- **TransferMsg:** Requests a fungible token transfer. Stores a commitment to the packet data and timeout in the source chain.
- **MsgRecvPacket:** Informs that a packet has been received and processed after verification of its commitment proof in the source chain. Stores a proof of packet acknowledgement in the destination chain.
- **MsgAcknowledgement:** Informs that a packet has been processed and acknowledged in the destination chain.

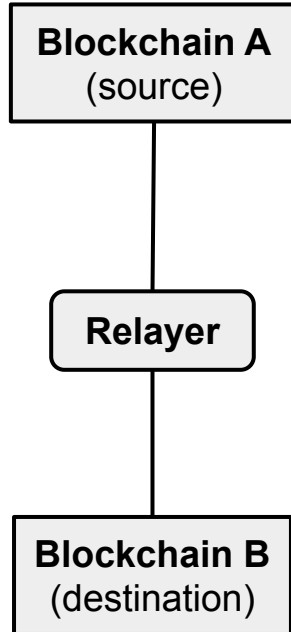
Fungible token transfer using IBC



Token transfers are composed by 3 steps:

- **TransferMsg:** Requests a fungible token transfer. Stores a commitment to the packet data and timeout in the source chain.
- **MsgRecvPacket:** Informs that a packet has been received and processed after verification of its commitment proof in the source chain. Stores a proof of packet acknowledgement in the destination chain.
- **MsgAcknowledgement:** Informs that a packet has been processed and acknowledged in the destination chain.

Fungible token transfer using IBC

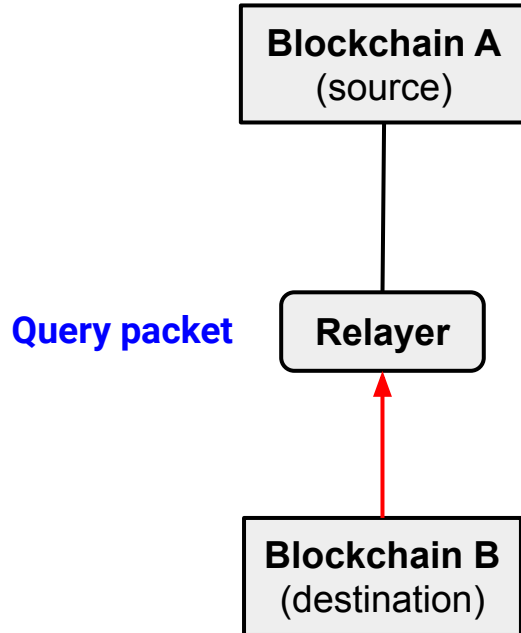


Commit MsgRecvPacket

Token transfers are composed by 3 steps:

- **TransferMsg:** Requests a fungible token transfer. Stores a commitment to the packet data and timeout in the source chain.
- **MsgRecvPacket:** Informs that a packet has been received and processed after verification of its commitment proof in the source chain. Stores a proof of packet acknowledgement in the destination chain.
- **MsgAcknowledgement:** Informs that a packet has been processed and acknowledged in the destination chain.

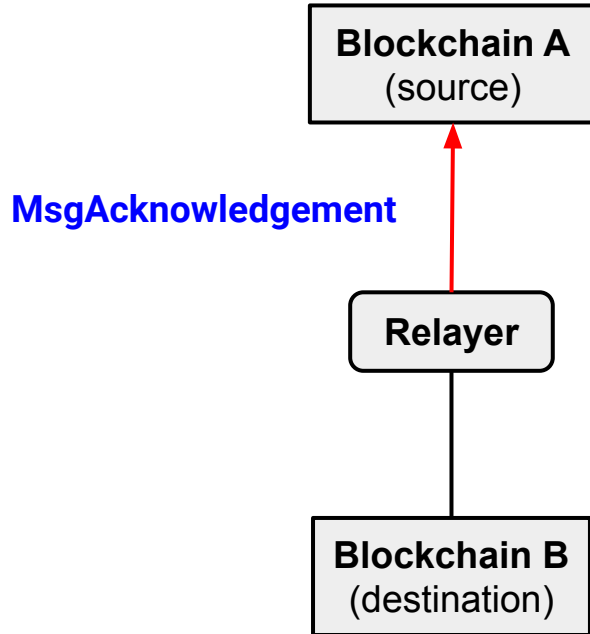
Fungible token transfer using IBC



Token transfers are composed by 3 steps:

- **TransferMsg:** Requests a fungible token transfer. Stores a commitment to the packet data and timeout in the source chain.
- **MsgRecvPacket:** Informs that a packet has been received and processed after verification of its commitment proof in the source chain. Stores a proof of packet acknowledgement in the destination chain.
- **MsgAcknowledgement:** Informs that a packet has been processed and acknowledged in the destination chain.

Fungible token transfer using IBC

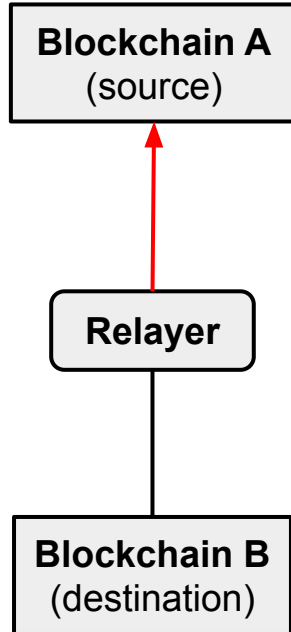


Token transfers are composed by 3 steps:

- **TransferMsg:** Requests a fungible token transfer. Stores a commitment to the packet data and timeout in the source chain.
- **MsgRecvPacket:** Informs that a packet has been received and processed after verification of its commitment proof in the source chain. Stores a proof of packet acknowledgement in the destination chain.
- **MsgAcknowledgement:** Informs that a packet has been processed and acknowledged in the destination chain.

Fungible token transfer using IBC

Commit
Acknowledgement



Token transfers are composed by 3 steps:

- **TransferMsg:** Requests a fungible token transfer. Stores a commitment to the packet data and timeout in the source chain.
- **MsgRecvPacket:** Informs that a packet has been received and processed after verification of its commitment proof in the source chain. Stores a proof of packet acknowledgement in the destination chain.
- **MsgAcknowledgement:** Informs that a packet has been processed and acknowledged in the destination chain.

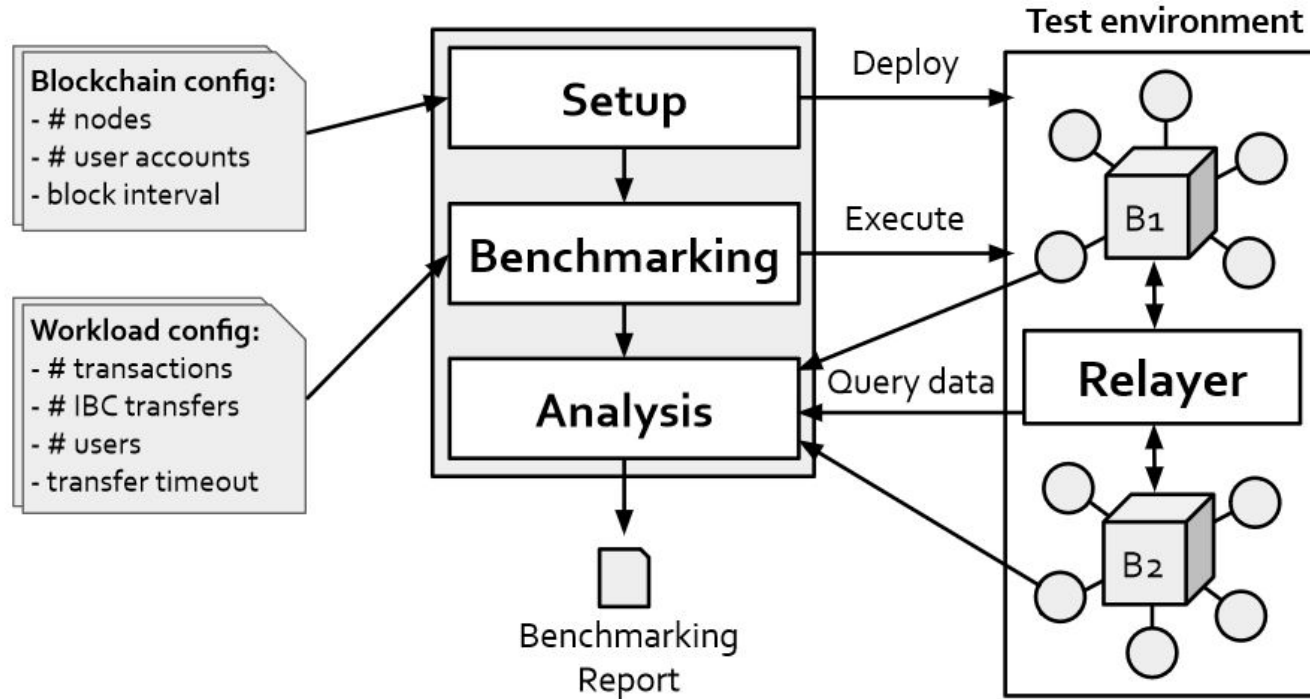
Our goal

- **Analyze the performance of the IBC protocol when used to send fungible token transfers between two Cosmos ecosystem blockchains:**
 - **Throughput:** measures cross-chain transfers completed per second (*transfer, recv, ack*)
 - **Latency:** measures the amount of time it takes for operations to be completed in seconds
 - **Relayer scalability:** measures change in throughput and latency according to the number of concurrent relayers in the same channel
- **Identify performance bottlenecks**
- **Identify issues/challenges regarding the deployment of cross-chain communications using IBC to help improve the protocol**

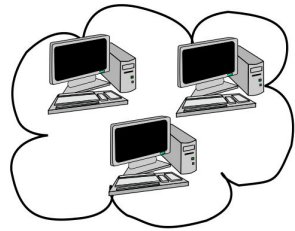


Measuring performance

- To measure IBC performance we developed a tool to automate deployment, workload execution and data analysis



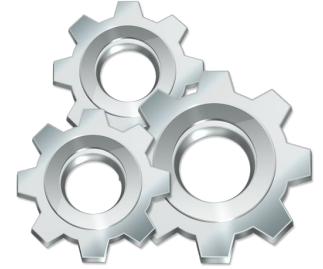
Deployment configuration



- **Private testnet environment:**

- 2 **Cosmos Gaia** v7.0.3 blockchains, each maintained by 5 validator nodes
- Ordered cross-chain channel established using the **Hermes Relay** v1.0.0
 - One of various relayers (also ibc-go, typescript relayer)
 - Larger community and more comprehensive documentation
 - More frequently updated, more discussions and issues on github
- 5 machines (i7-9700 3GHz, 16GB 2666 MT/s RAM, 7200RPM HDDs, Debian 11) in a LAN environment. WAN conditions simulated by enforcing **200ms** round-trip latency
- The Hermes Relay application is connected to two full nodes through local endpoints

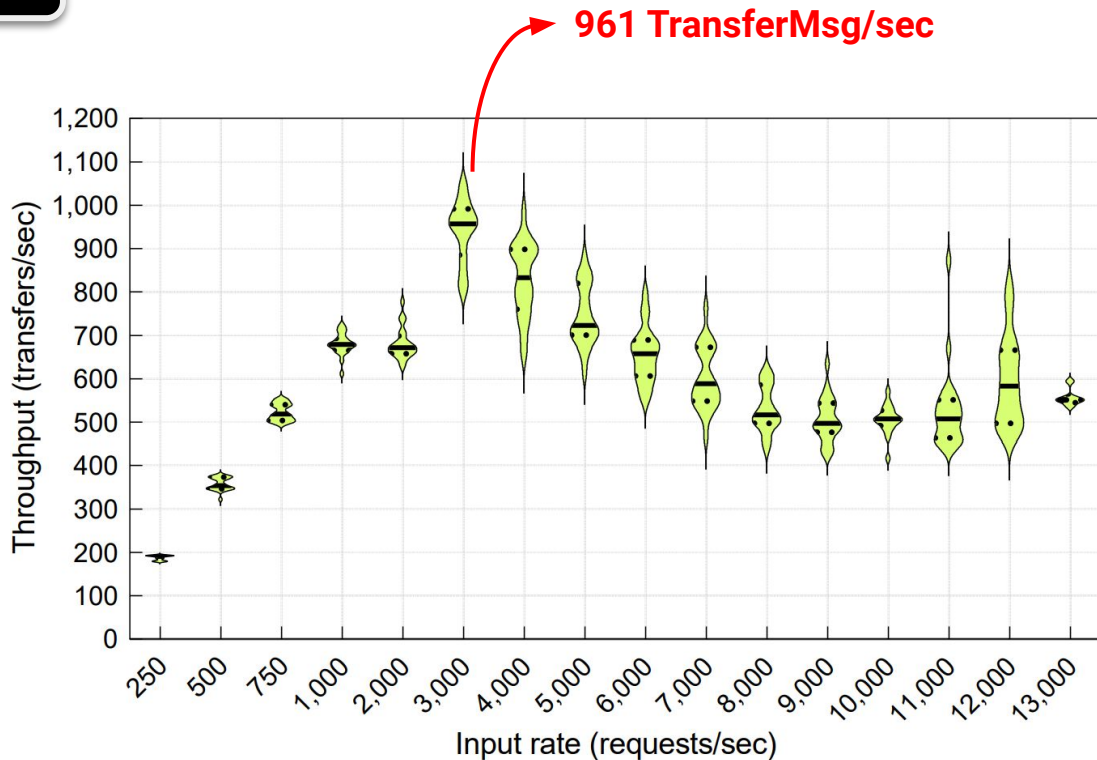
Experiment settings



- **Setup:**
 - Blockchains started from genesis for every execution
 - Cross-chain channel established for every execution
- **Tendermint:**
 - Block interval of 5 seconds (lower bound)
- **Workload:**
 - Submitted transactions contain 100 (max) transfer requests to stress the relayer
 - Multiple users submit transactions concurrently to overcome Cosmos SDK limitation on number of submitted transactions

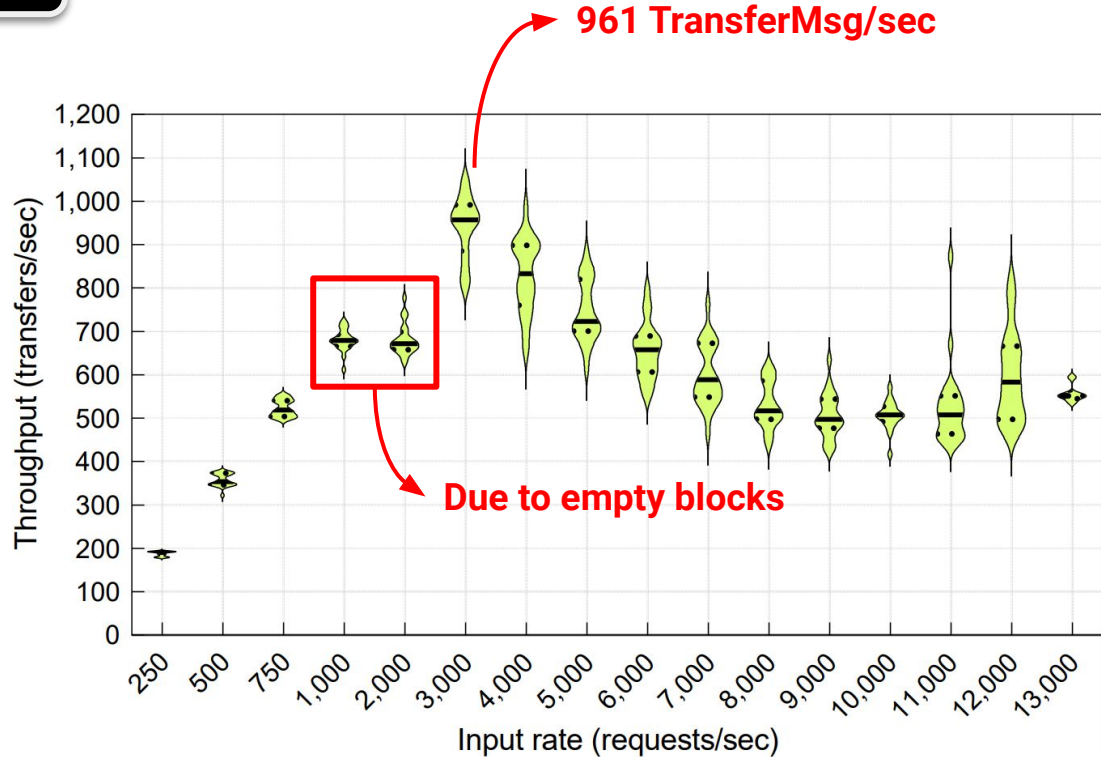
Tendermint throughput

- Tendermint **TransferMsg** throughput capacity (20 execs each)
- From 250 to 14,000 transfer requests per second
- Cross-chain relaying disabled
- **961 TransferMsg/sec peak throughput**
- Roughly **10x** more transfers than the relayer is able to complete



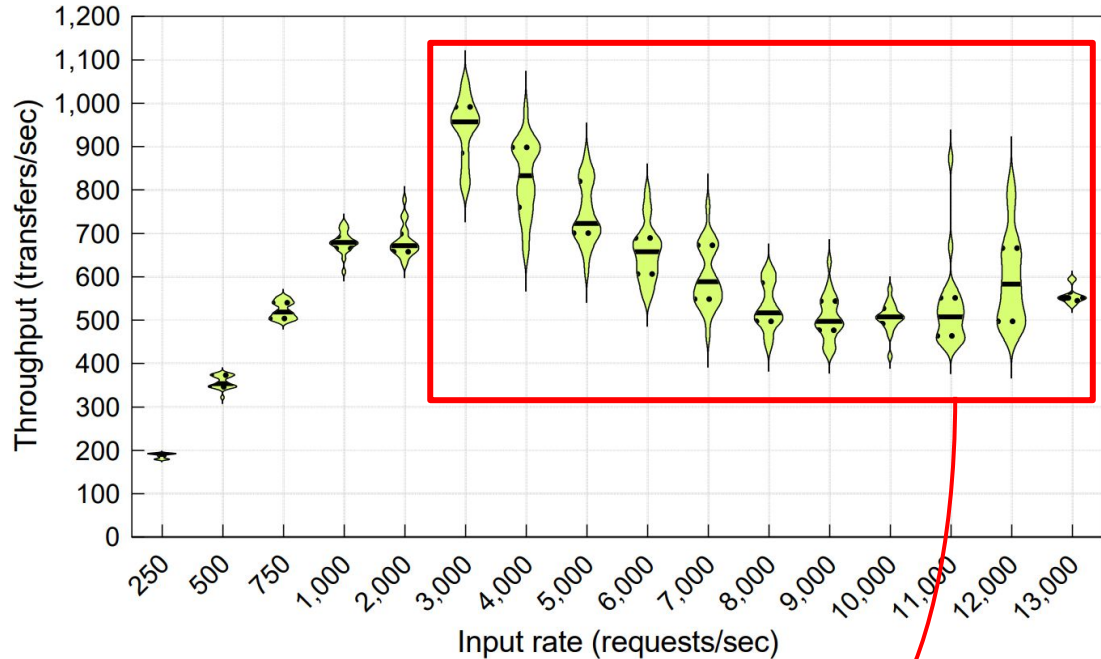
Tendermint throughput

- Tendermint **TransferMsg** throughput capacity (20 execs each)
- From 250 to 14,000 transfer requests per second
- Cross-chain relaying disabled
- **961 TransferMsg/sec peak throughput**
- Roughly **10x** more transfers than the relayer is able to complete



Tendermint throughput

- Tendermint **TransferMsg** throughput capacity (20 execs each)
- From 250 to 14,000 transfer requests per second
- Cross-chain relaying disabled
- **961 TransferMsg/sec peak throughput**
- Roughly **10x** more transfers than the relayer is able to complete



Inconsistent results past peak throughput:

"failed tx: no confirmation"

"account sequence mismatch"

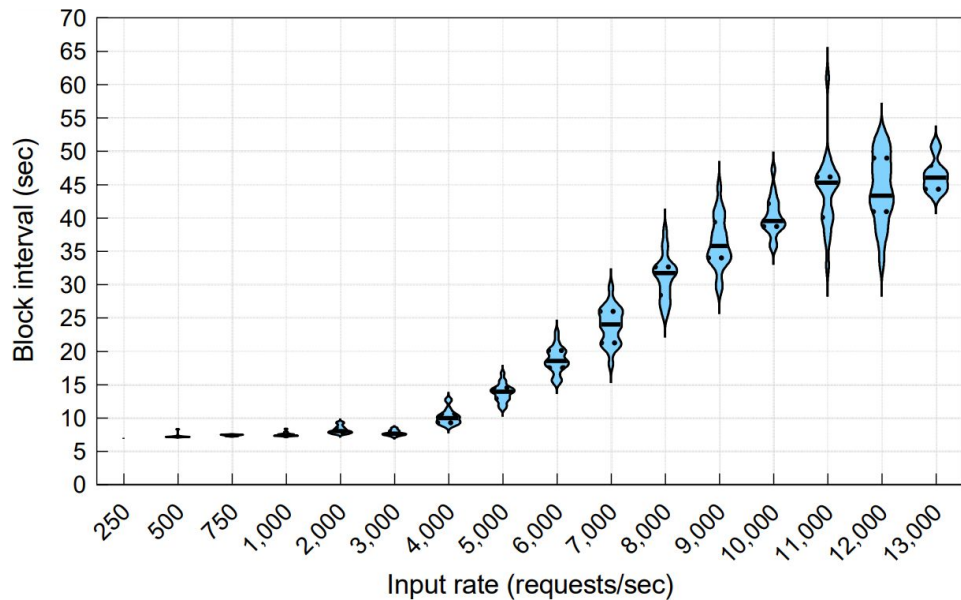
Transfer failure rate

<i>Input rate (requests/sec)</i>	<i>Requests made to Hermes</i>	<i>Submitted to blockchain</i>	<i>Committed (from submitted)</i>
250 to 9,000	18,750 to 675,000	>99%	>99%
10,000	750,000	601,300 (80.17%)	591,450 (98.3%)
11,000	825,000	319,152 (38.6%)	292,424 (91.6%)
12,000	900,000	160,343 (17.8%)	119,733 (74.6%)
13,000	975,000	100,688 (10.3%)	51,436 (51%)
14,000	1,050,000	90,000 (8.5%)	26,360 (29.2%)

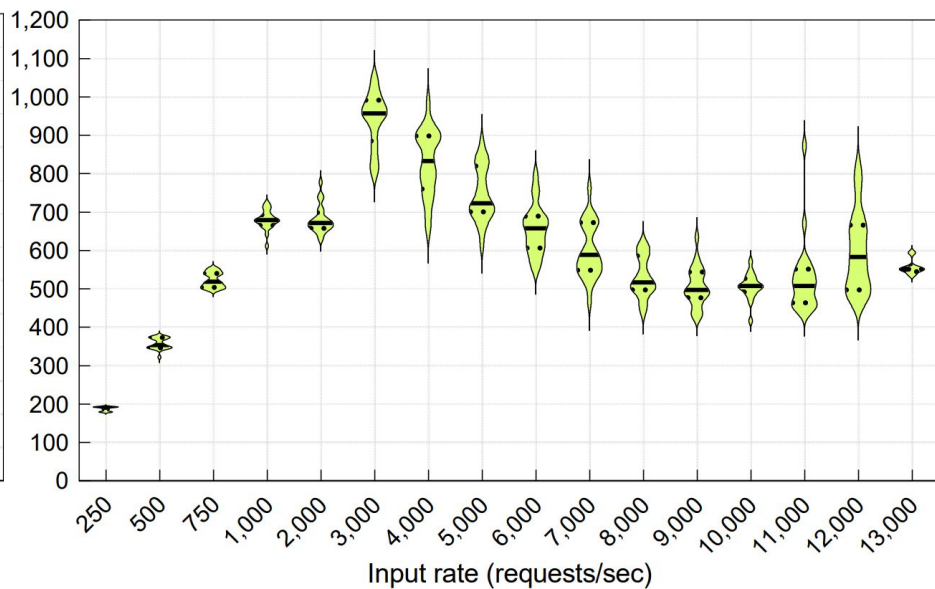
High req/s stresses the RPC endpoint, which is also used to query for confirmed transactions and increase account sequence numbers

Block interval vs. throughput

Avg. block interval

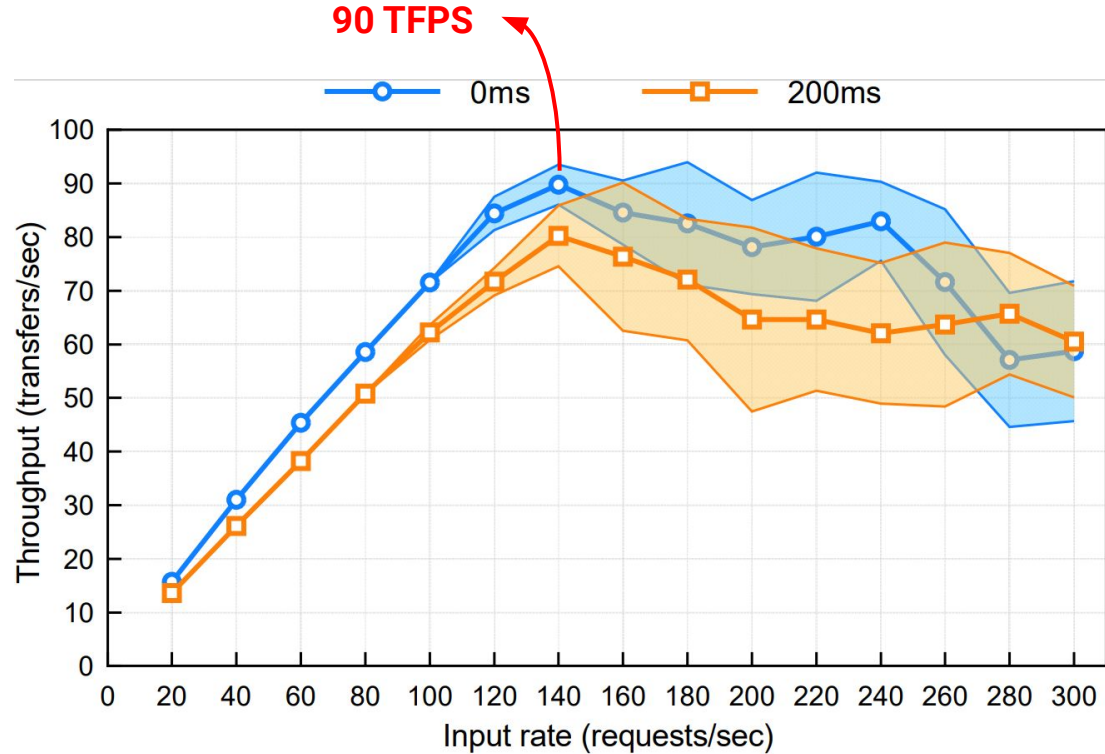


Tendermint throughput



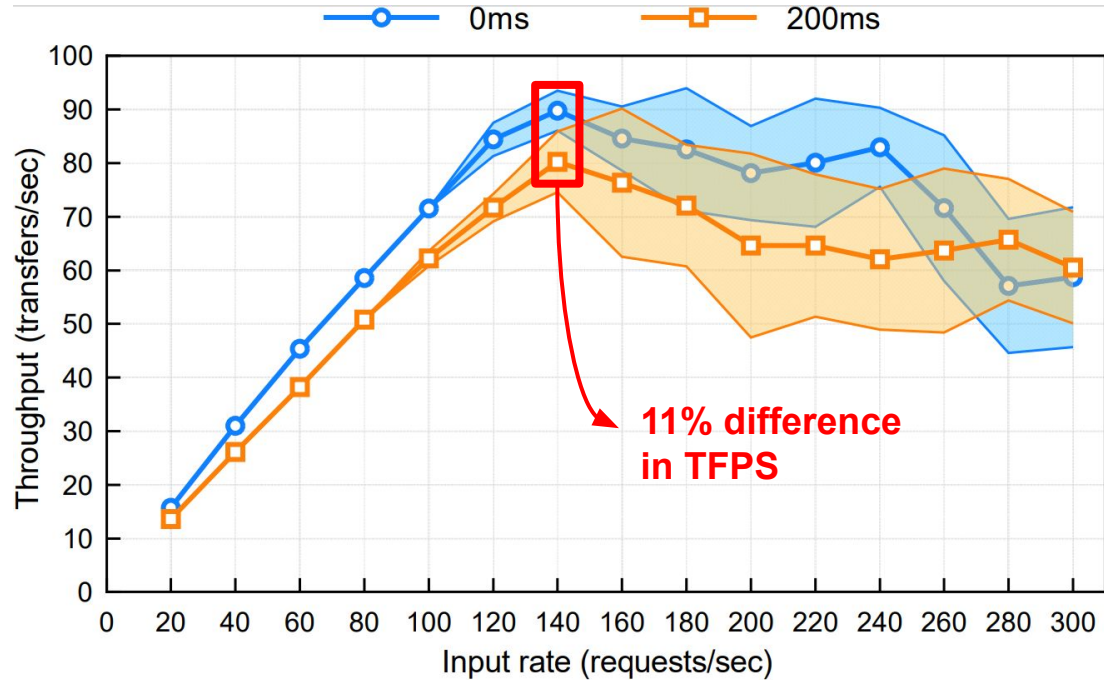
Hermes Relay - 1 Relay

- Cross-chain throughput (*transfer, recv, ack*) for **50 consecutive blocks**, 20 executions for each data point
- From 20 to 300 transfer requests per second
- Both 0ms and 200ms latency
- Error bands depict standard deviation
- 80 cross-chain transfers/sec peak throughput with 200ms latency



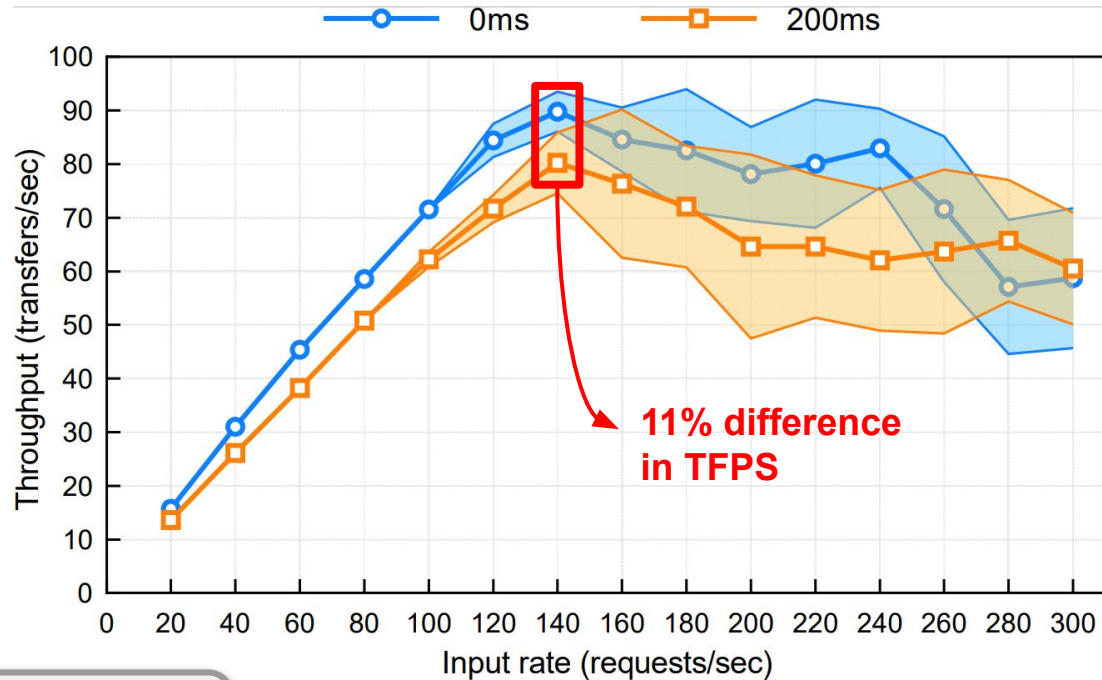
Hermes Relay - 1 Relay

- Cross-chain throughput (*transfer, recv, ack*) for **50 consecutive blocks**, 20 executions for each data point
- From 20 to 300 transfer requests per second
- Both 0ms and 200ms latency
- Error bands depict standard deviation
- 80 cross-chain transfers/sec peak throughput with 200ms latency



Hermes Relay - 1 Relay

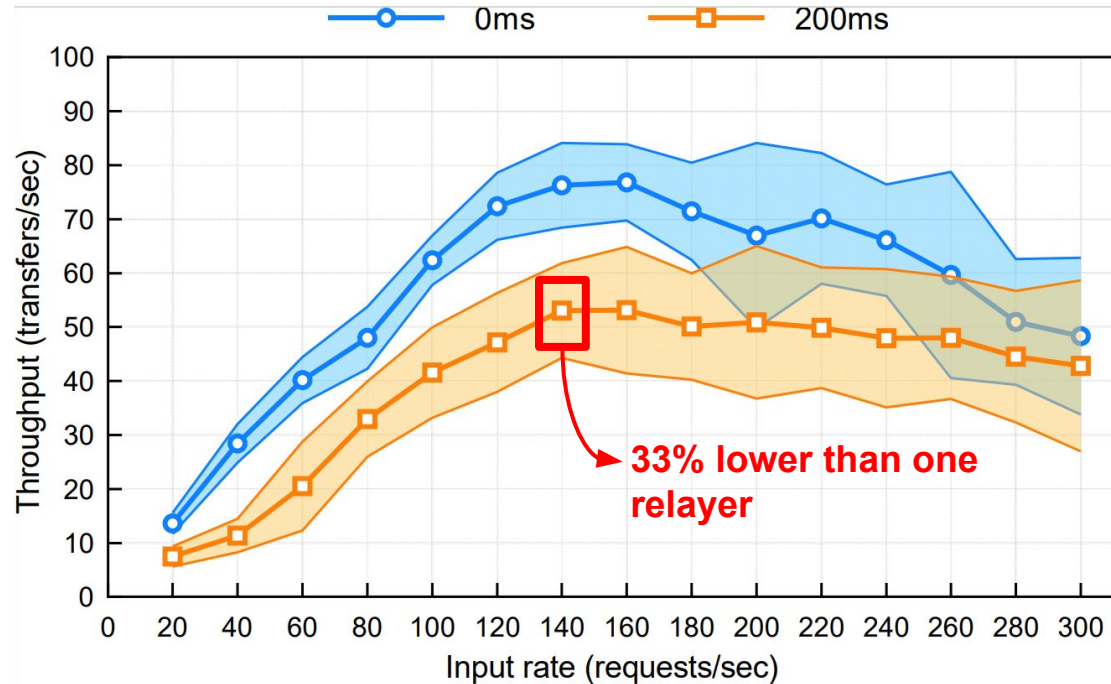
- Cross-chain throughput (*transfer, recv, ack*) for **50 consecutive blocks**, 20 executions for each data point
- From 20 to 300 transfer requests per second
- Both 0ms and 200ms latency
- Error bands depict standard deviation
- 80 cross-chain transfers/sec peak throughput with 200ms latency



Can we increase throughput by increasing the number of relayers?

Hermes Relay - 2 Relayers

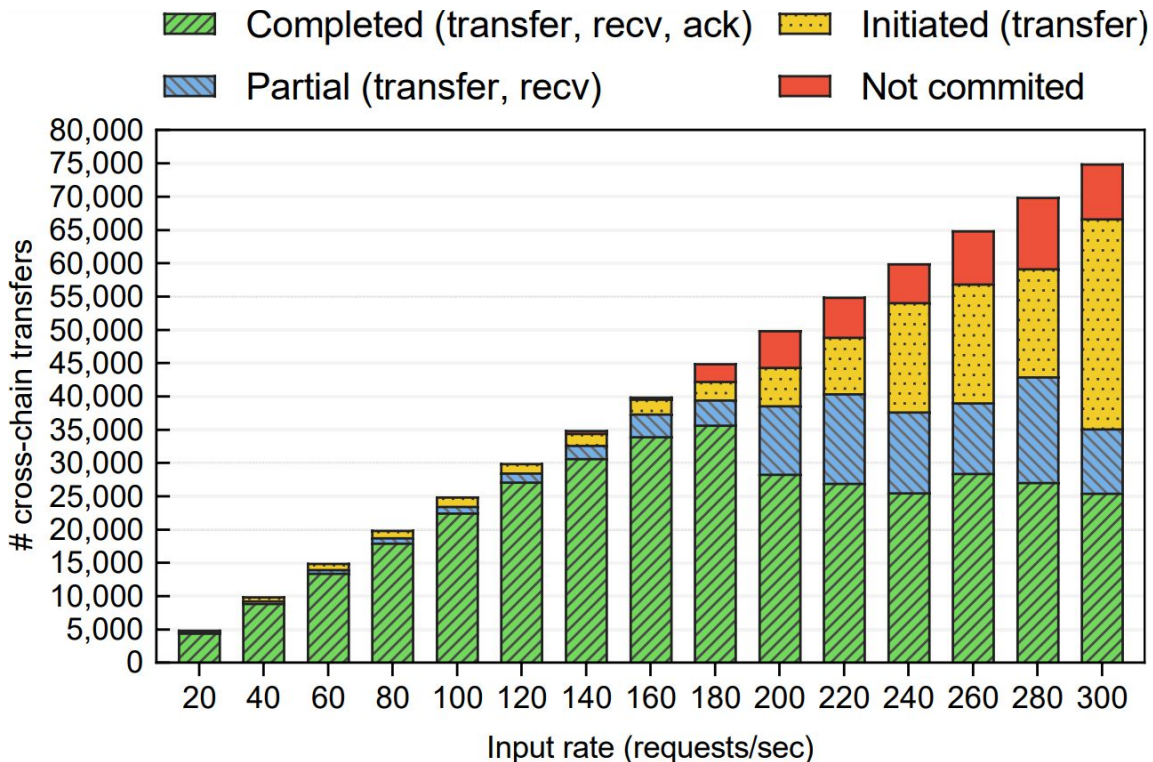
- **Peak throughput:**
 - 77 TFPS with 0ms
 - 53 TFPS with 200ms
- **33% lower throughput** for 200ms compared to one relayer. Why?
- *Several "Packet messages are redundant" errors, 23k for 100 RPS*
- No coordination between relayers in the same channel
- **What about two different channels?**
 - Different denominations, i.e, non-fungible



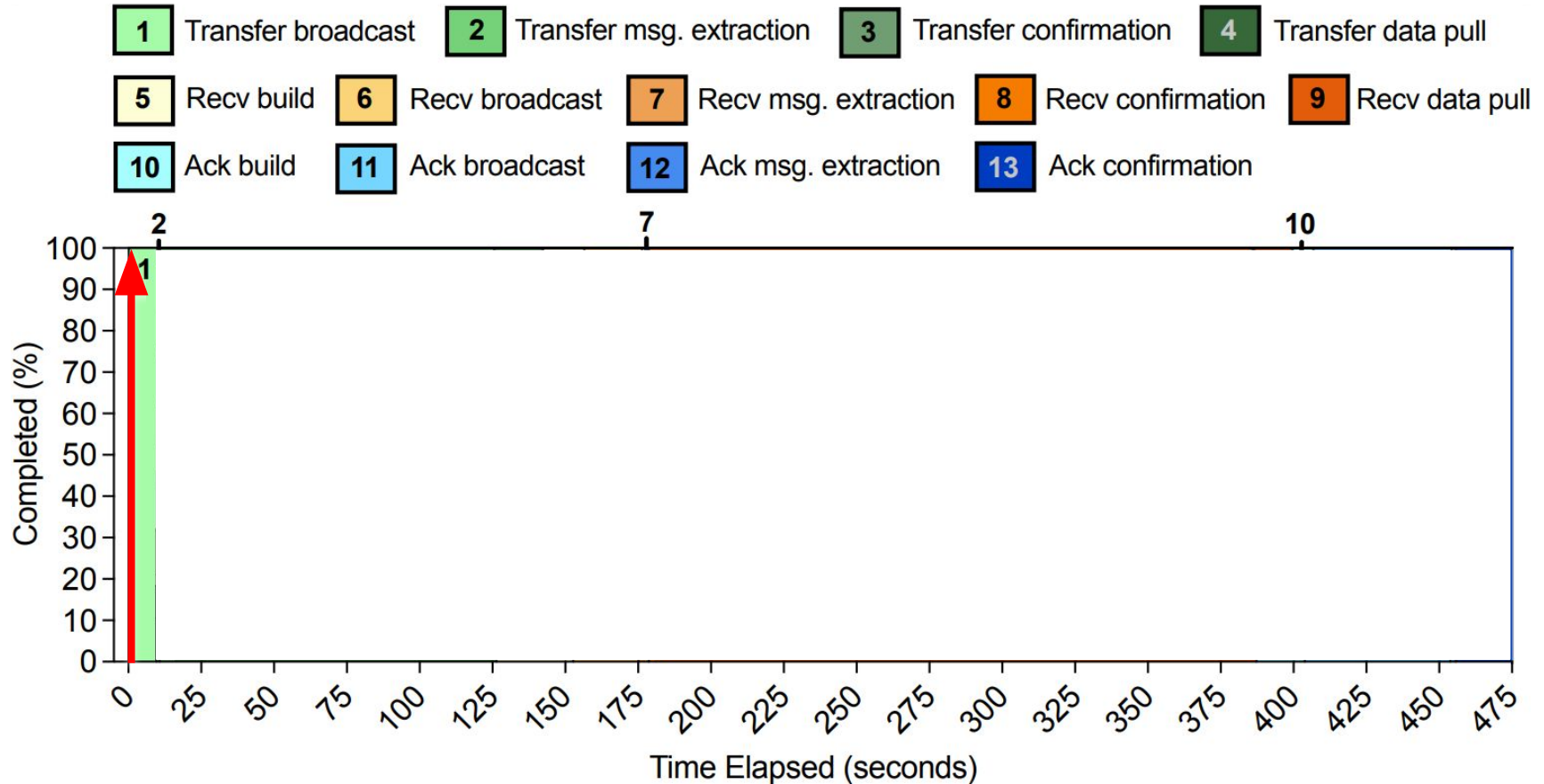
Lower performance but maintains liveness

IBC Messages committed - 1 Relay

- Less transfers completed with larger workloads (more pending messages)
- Transactions are completed after the experiment interval (50 blocks)
- Why do transfers take so long to complete?
 - Look at completion latency next

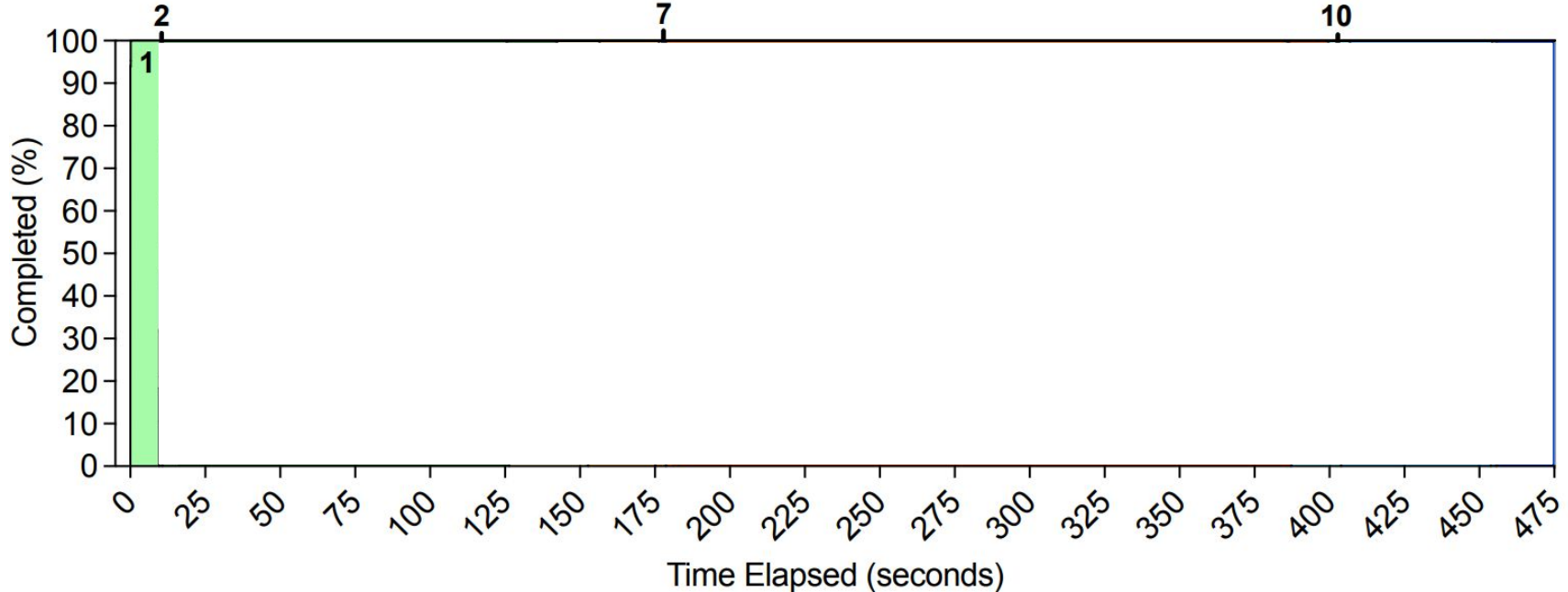


Operation latency (5k transfers)



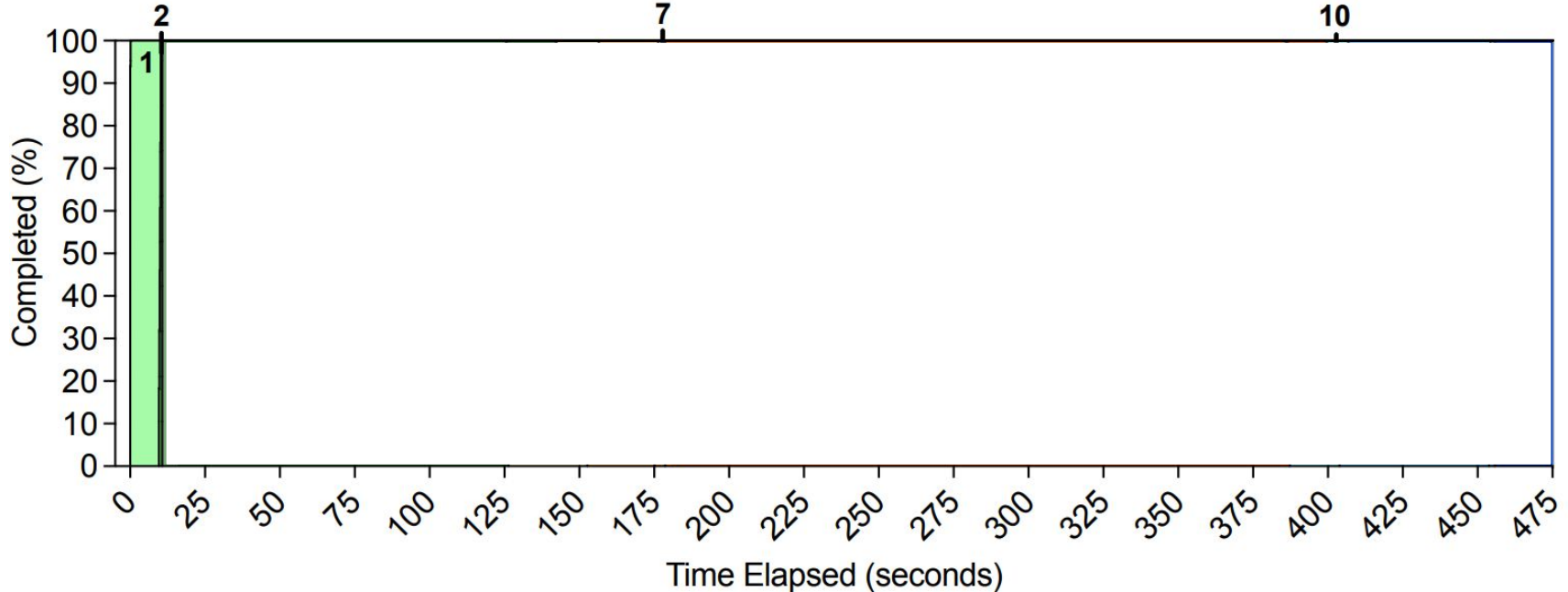
Operation latency (5k transfers)

- 1 Transfer broadcast
- 2 Transfer msg. extraction
- 3 Transfer confirmation
- 4 Transfer data pull
- 5 Recv build
- 6 Recv broadcast
- 7 Recv msg. extraction
- 8 Recv confirmation
- 9 Recv data pull
- 10 Ack build
- 11 Ack broadcast
- 12 Ack msg. extraction
- 13 Ack confirmation



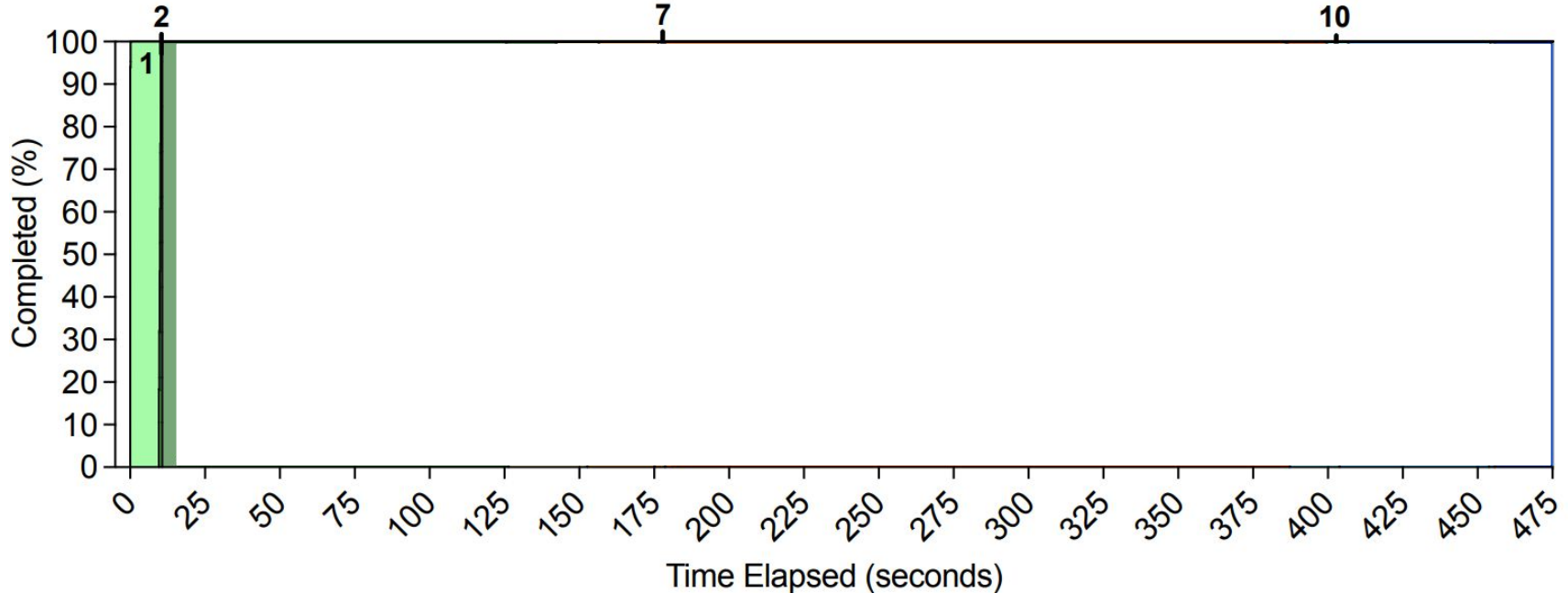
Operation latency (5k transfers)

- 1 Transfer broadcast
- 2 Transfer msg. extraction
- 3 Transfer confirmation
- 4 Transfer data pull
- 5 Recv build
- 6 Recv broadcast
- 7 Recv msg. extraction
- 8 Recv confirmation
- 9 Recv data pull
- 10 Ack build
- 11 Ack broadcast
- 12 Ack msg. extraction
- 13 Ack confirmation



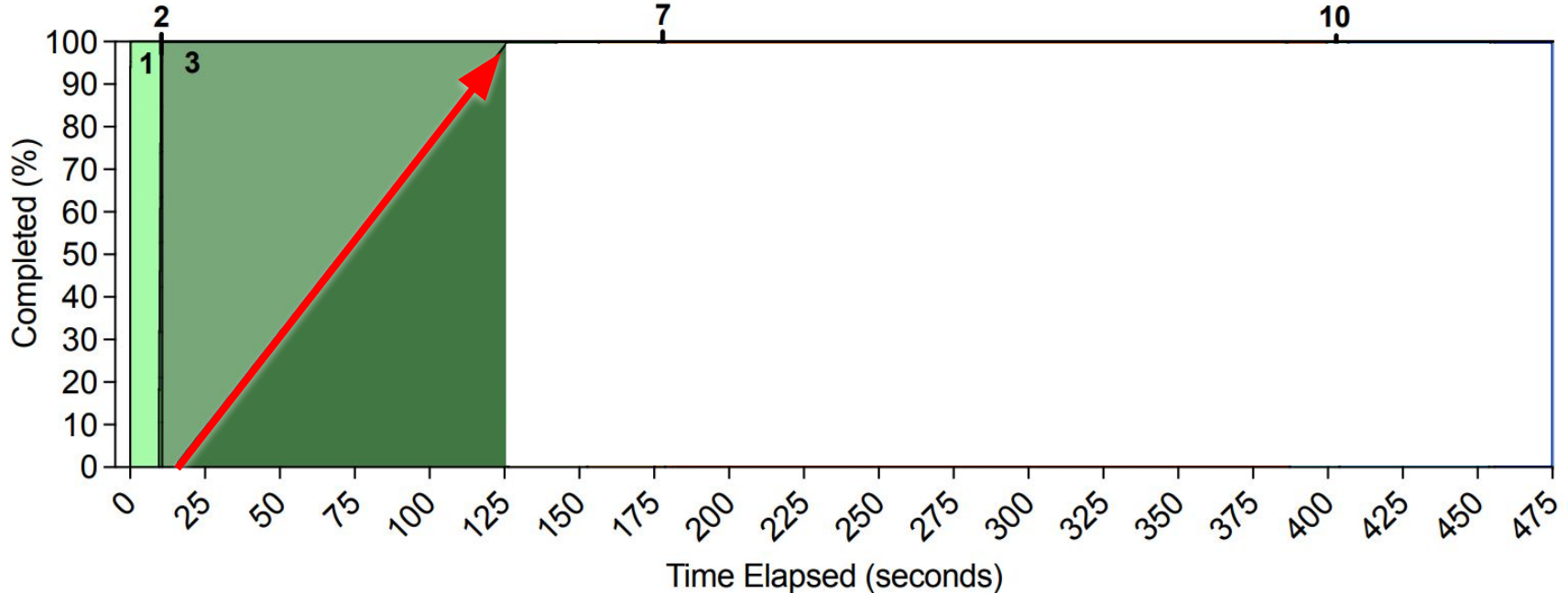
Operation latency (5k transfers)

- 1 Transfer broadcast
- 2 Transfer msg. extraction
- 3 Transfer confirmation
- 4 Transfer data pull
- 5 Recv build
- 6 Recv broadcast
- 7 Recv msg. extraction
- 8 Recv confirmation
- 9 Recv data pull
- 10 Ack build
- 11 Ack broadcast
- 12 Ack msg. extraction
- 13 Ack confirmation



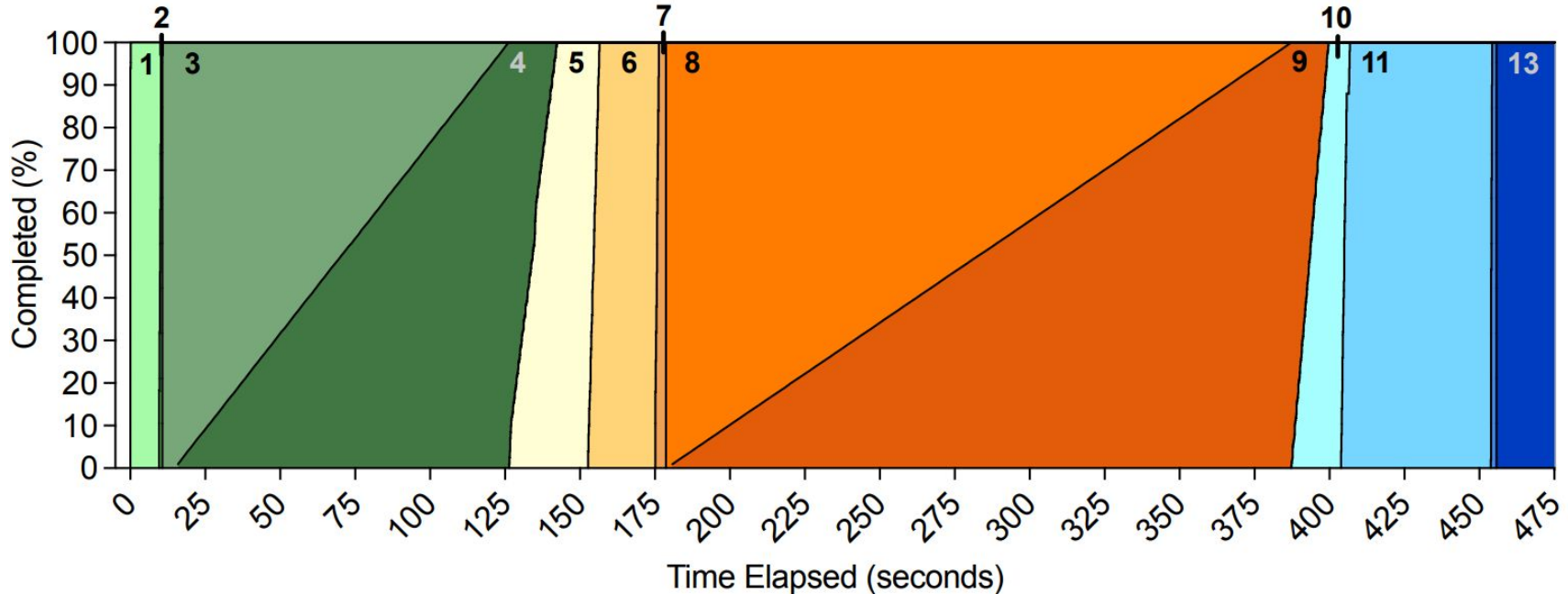
Operation latency (5k transfers)

- 1 Transfer broadcast
- 2 Transfer msg. extraction
- 3 Transfer confirmation
- 4 Transfer data pull
- 5 Recv build
- 6 Recv broadcast
- 7 Recv msg. extraction
- 8 Recv confirmation
- 9 Recv data pull
- 10 Ack build
- 11 Ack broadcast
- 12 Ack msg. extraction
- 13 Ack confirmation



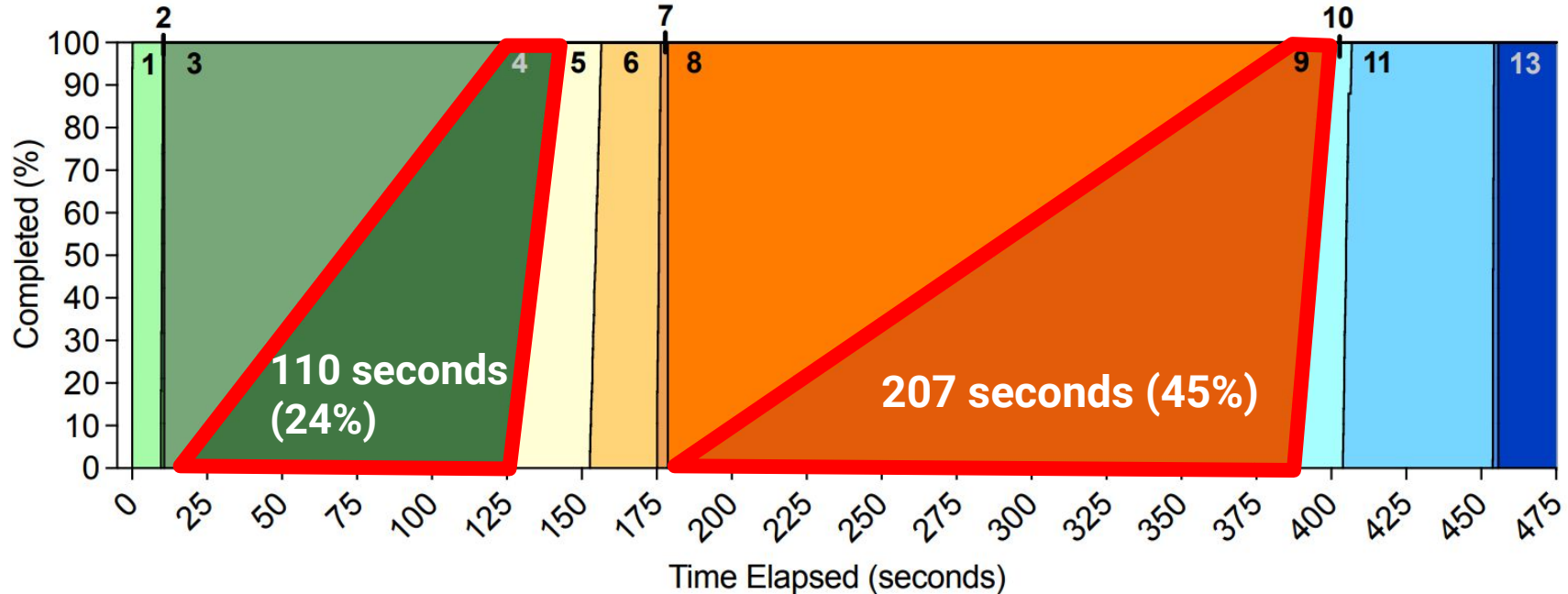
Operation latency (5k transfers)

- 1 Transfer broadcast
- 2 Transfer msg. extraction
- 3 Transfer confirmation
- 4 Transfer data pull
- 5 Recv build
- 6 Recv broadcast
- 7 Recv msg. extraction
- 8 Recv confirmation
- 9 Recv data pull
- 10 Ack build
- 11 Ack broadcast
- 12 Ack msg. extraction
- 13 Ack confirmation



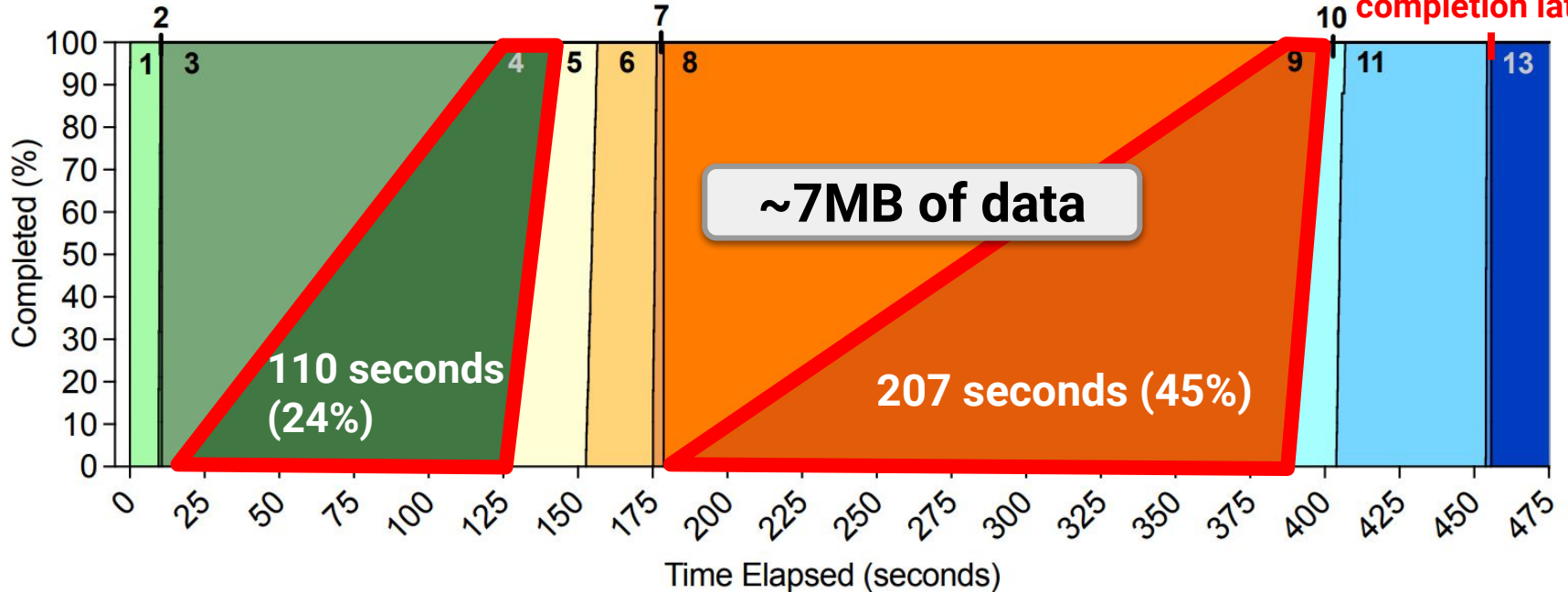
Operation latency (5k transfers)

- 1 Transfer broadcast
- 2 Transfer msg. extraction
- 3 Transfer confirmation
- 4 Transfer data pull
- 5 Recv build
- 6 Recv broadcast
- 7 Recv msg. extraction
- 8 Recv confirmation
- 9 Recv data pull
- 10 Ack build
- 11 Ack broadcast
- 12 Ack msg. extraction
- 13 Ack confirmation



Operation latency (5k transfers)

- 1 Transfer broadcast
- 2 Transfer msg. extraction
- 3 Transfer confirmation
- 4 Transfer data pull
- 5 Recv build
- 6 Recv broadcast
- 7 Recv msg. extraction
- 8 Recv confirmation
- 9 Recv data pull
- 10 Ack build
- 11 Ack broadcast
- 12 Ack msg. extraction
- 13 Ack confirmation



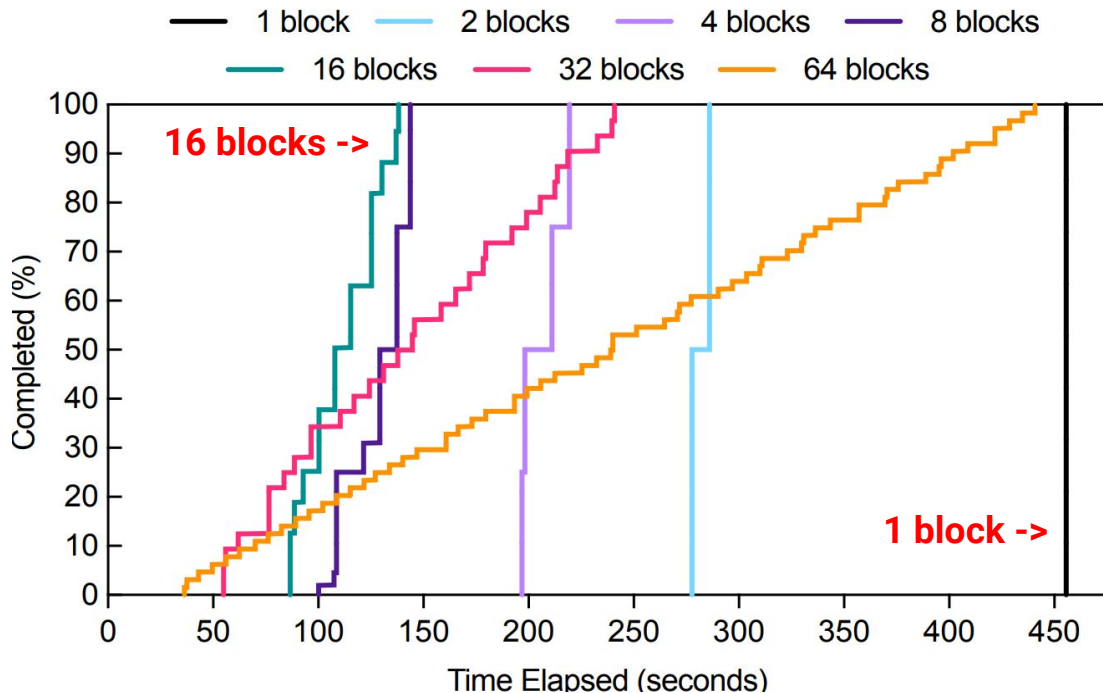
Completion latency (5k transfers)

- Can we improve completion latency? What's the best strategy?

- Divide 5k transfer submissions across increasing number of blocks

- **1 block:** 455 sec
- **2 blocks:** 286 sec ↓
- **4 blocks:** 219 sec ↓
- **8 blocks:** 143 sec ↓
- **16 blocks:** 138 sec ↓
- **32 blocks:** 240 sec ↑
- **64 blocks:** 441 sec ↑

- Submission in large batches is easier but severely increases completion latency



70% reduction from 1 block (455s) to 16 blocks (138s)

Deployment challenges

- **Timestamp mismatch:** Events registered by the blockchain are “in the past” compared to the relayer, e.g, transaction committed before being broadcast
- **Account sequence mismatch:** Unable to submit transactions sequentially within a single block, requires transfers to be accumulated or multiple user accounts
- **Websocket space limit:** The Tendermint websocket has a maximum message size of 16MB. Blocks with more than 16MB crash the relayer as it tries to collect all events at once.
- **Transaction data collection:** The endpoints offer no query to retrieve only tx hashes. Queries return substantial amount of data and slow data analysis (579,919 lines of output and 5.7 seconds for 20 txs with 100 MsgRecvPacket each)
- **Incomplete logging for blockchain data retrieval:** Only a fraction of the data pull operations are recorded in the relayer’s logs if transactions span many blocks. Only data retrieval from the first block is recorded.

Contribution summary

- Provided an analysis on the performance of the IBC protocol and **identified bottlenecks** that impair relayer performance and lead to high transaction confirmation latency
- Developed an open source tool to facilitate performance measurement of cross-chain communication using Cosmos and the IBC protocol
- Identified challenges/issues in the process of deploying and using the relayer and the IBC protocol
- Generated a dataset with 158GB of execution logs to aid in future research

Thank you!

Email: joao.massarichervinski@monash.edu