#gigablocktestnet

Measuring maximum sustained transaction throughput on a global network of Bitcoin nodes

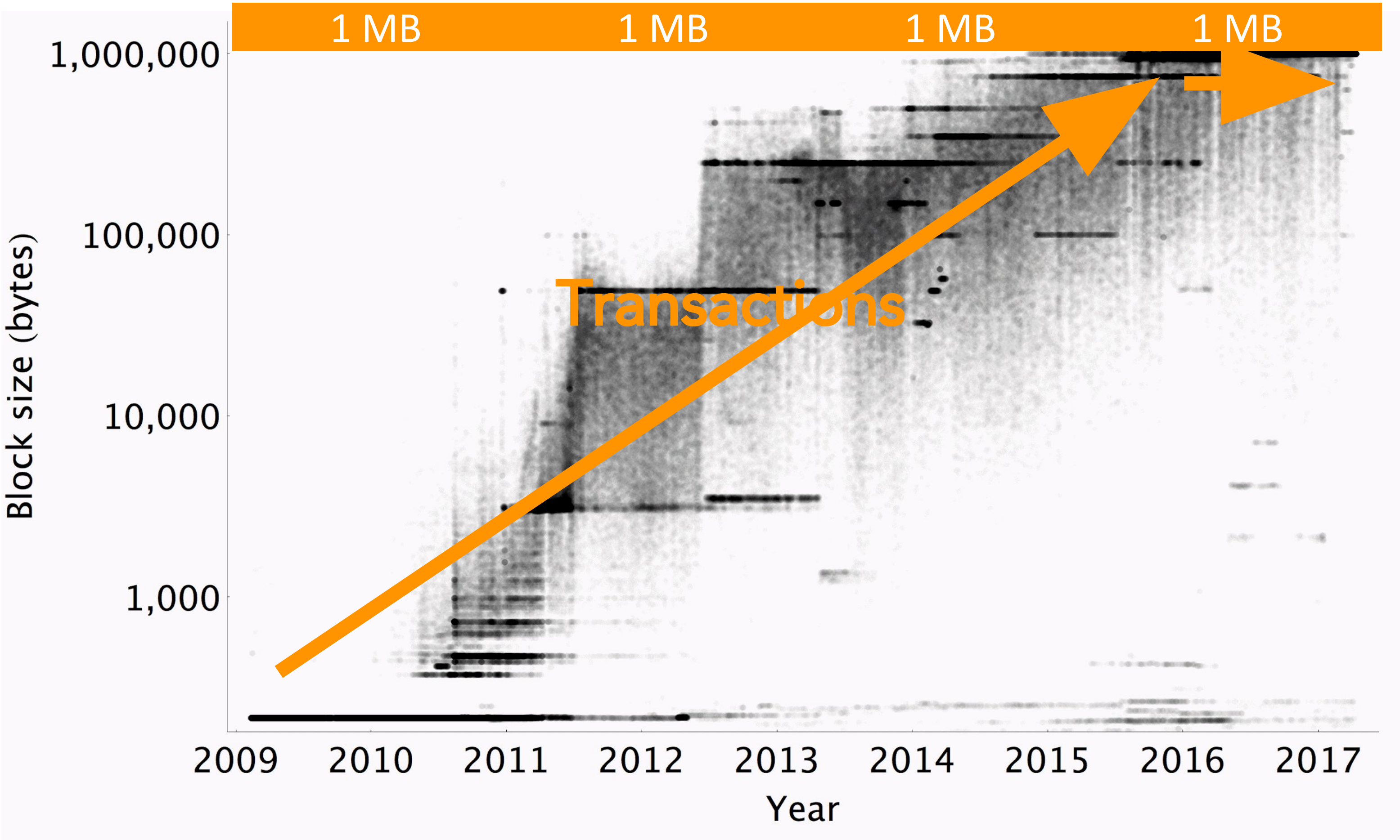
4 November 2017

Andrea Suisani,[1] Andrew Clifford,[1] Andrew Stone,[1] Erik Beijnoff,[1] Peter Rizun,[1] Peter Tschipper,[1] Alexandra Fedorova,[2] Chen Feng,[2] Victoria Lemieux,[2] Stefan Matthews[3]

[1] Bitcoin Unlimited, [2] University of British Columbia, [3] nChain

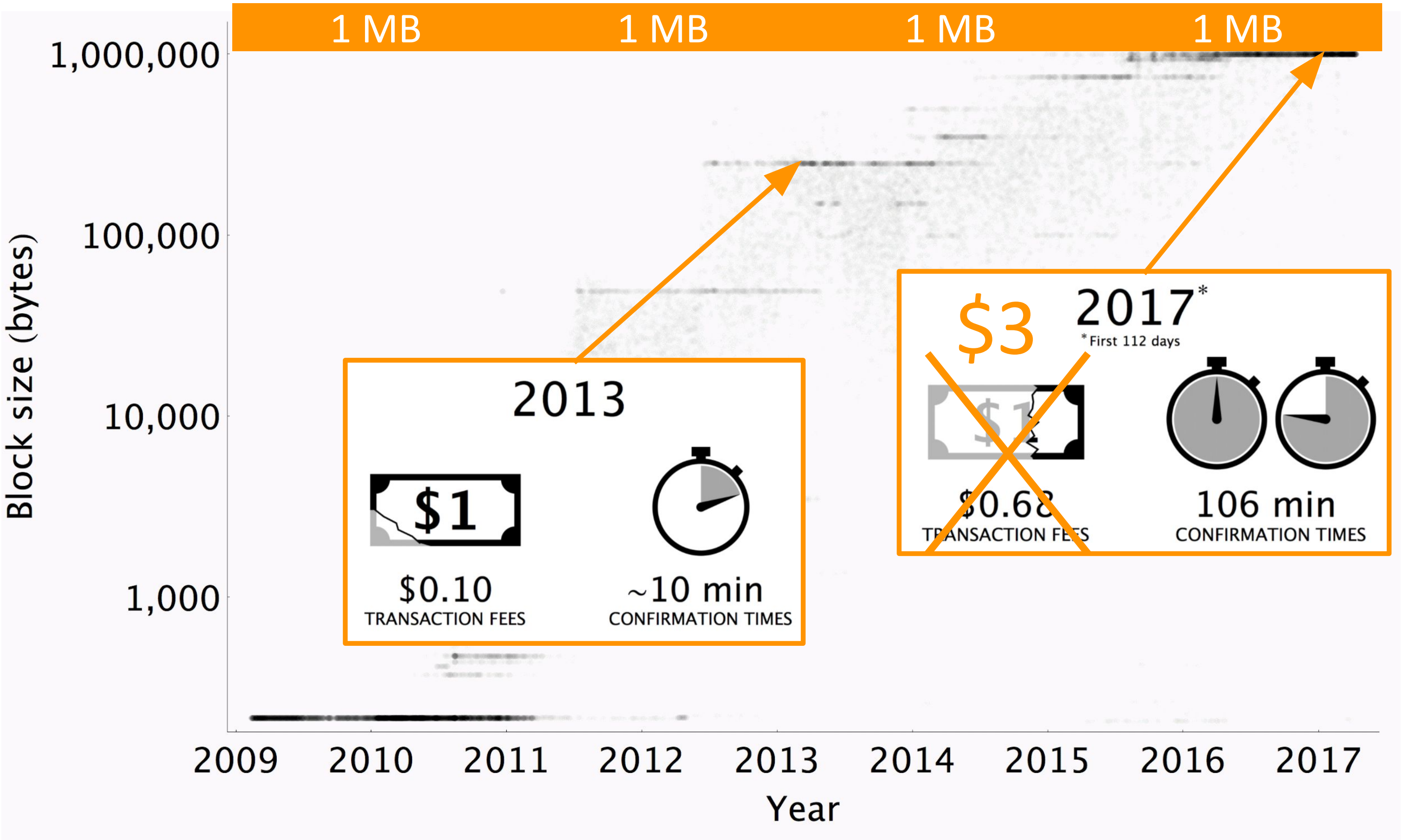
Motivation

- Transaction volume was growing exponentially
- Hitting the “1 MB block size limit” put a lid on growth



Motivation

- Transaction volume was growing exponentially
- Hitting the "1 MB block size limit" put a lid on growth
- Fees have increased and confirmation times have become unreliable
- We want to raise the limit but there are scaling concerns



Scaling concerns

USERS	NETWORK NODES

Scaling concerns

USERS

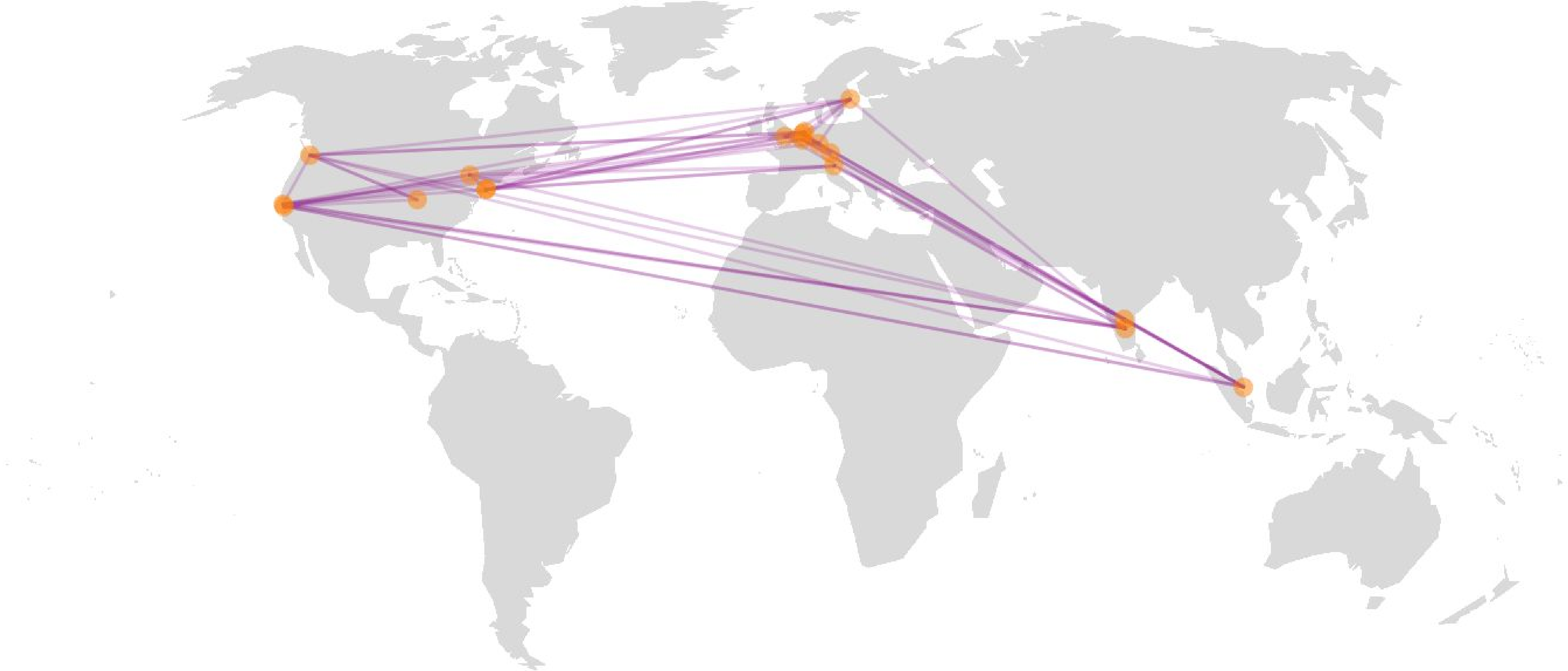
- Simplified payment verification (SPV) technology is highly scalable
- Users can:
  + Be there own banks
  + Verify their own transactions
  + Send payments to any other user
- 4 billion people already have access to technology to facilitate this ("dumb phone" + SMS text message)

NETWORK NODES

- Network nodes must validate every transaction
- 4 billion users x 1 transaction per day = **50,000 tx/sec**
- Network nodes are needed for:
  + Mining new blocks
  + Serving Merkle-branch proofs to SPV wallets
  + Archiving historical blocks
  + Some businesses (e.g., payment processing)
  + Research/development

We wanted to measure the maximum sustained throughput of a global network of bitcoin nodes to see how close we are to achieving this, and then to identify bottlenecks.

Gigablock Testnet (October 2017 – 18 nodes)



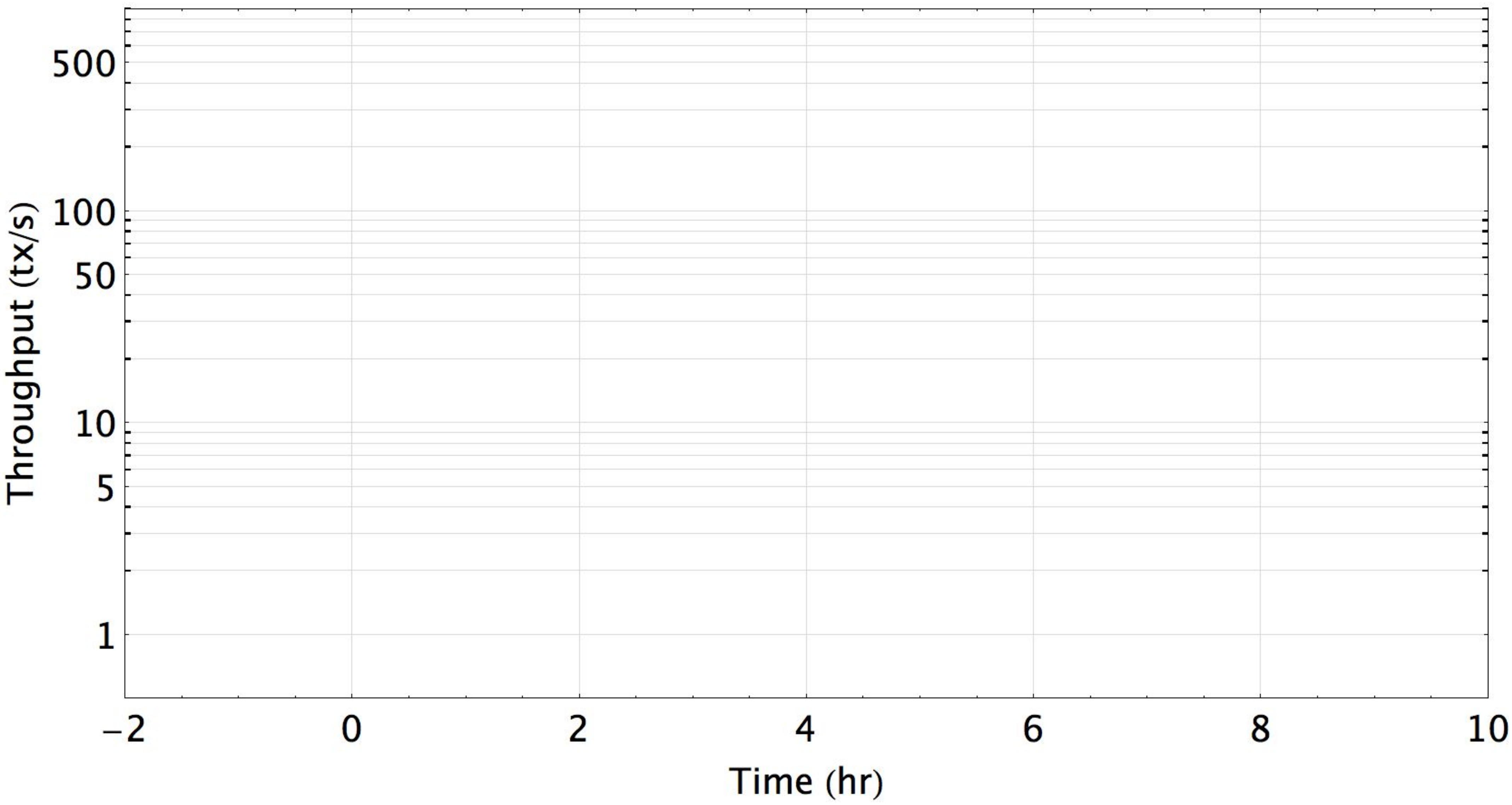
Gigablock Testnet (October 2017 – 18 nodes)

Reference spec

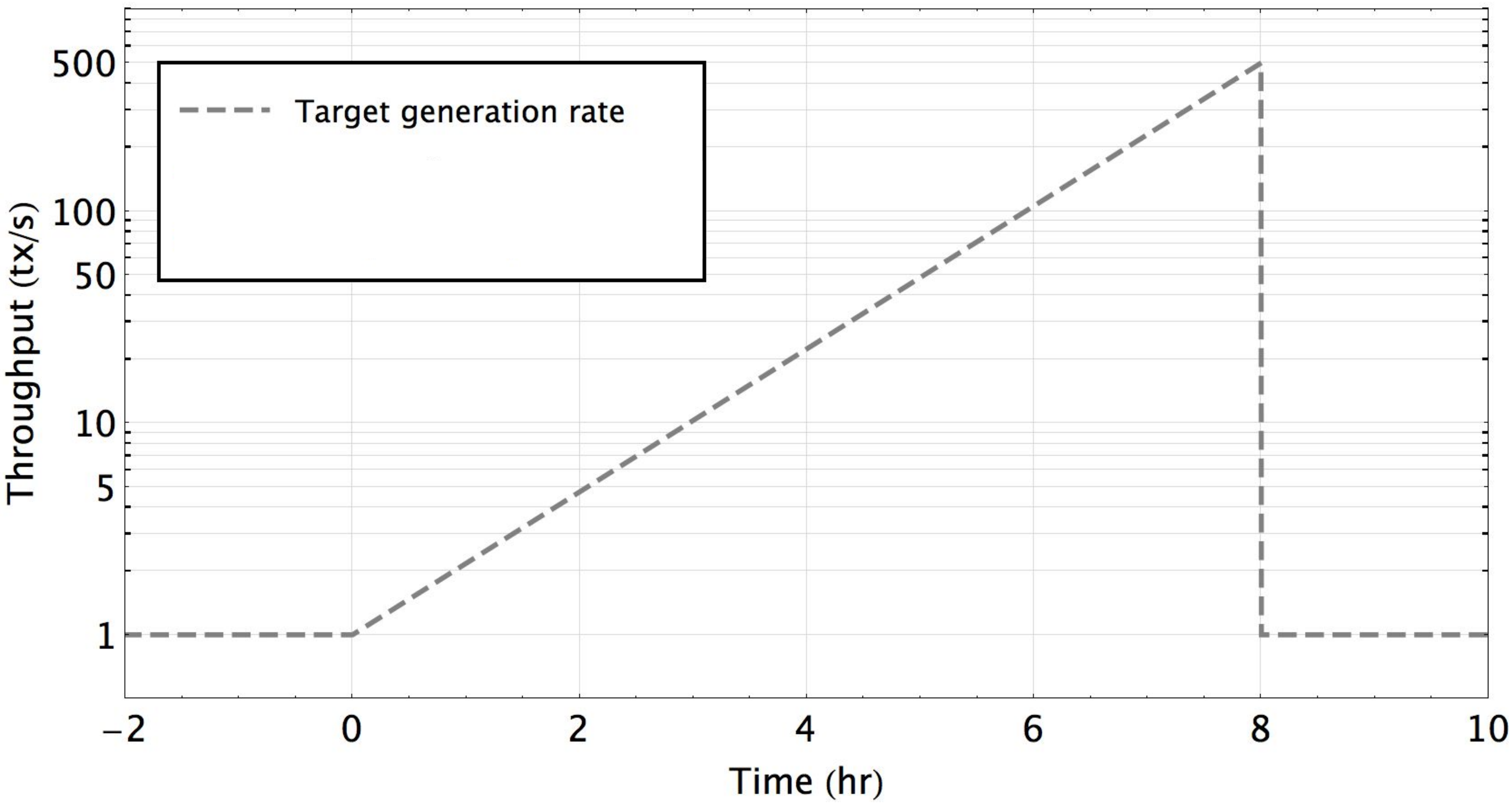
- 4-core CPU
- 30 Mbps
- 16 GB RAM
- SSD storage

- 4-6 miners — CPU mining
- 12 generators — Python scripts 2-in/2-out TXs

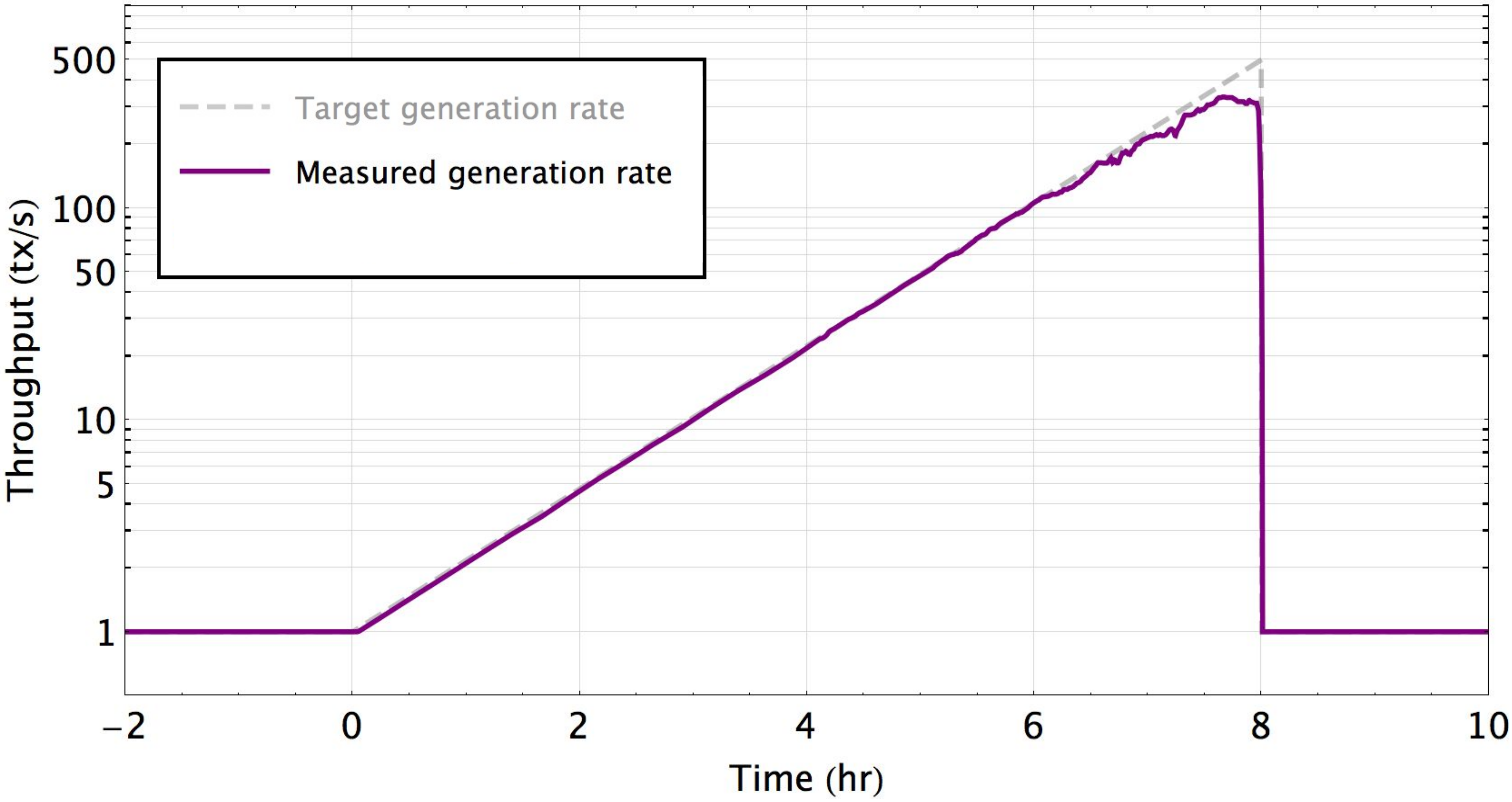
Ramp tests



Ramp tests

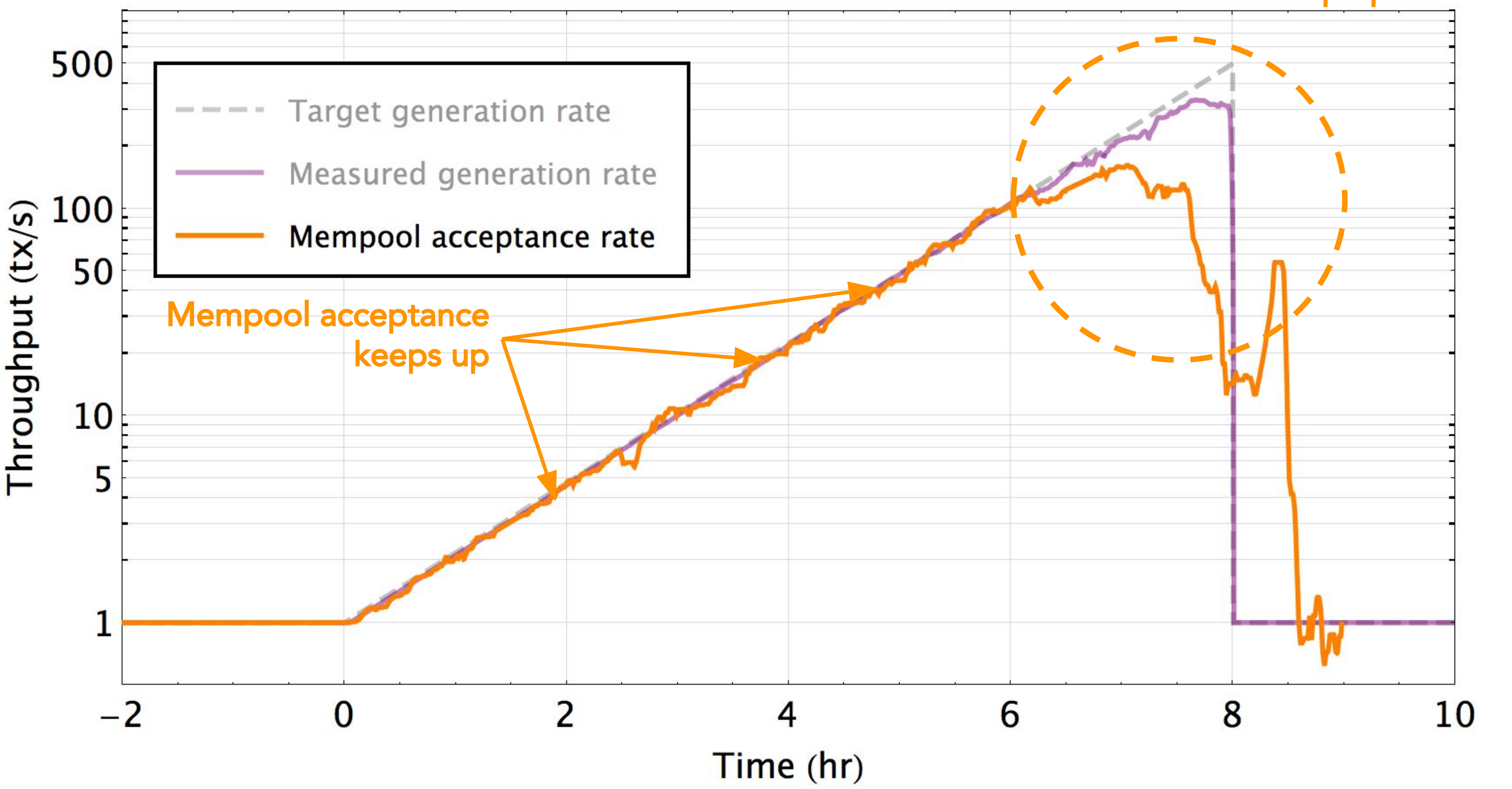

Target generation rate
Throughput (tx/s)
500
100
50
10
5
1
–2
0
2
4
6
8
10
Time (hr)

Ramp tests

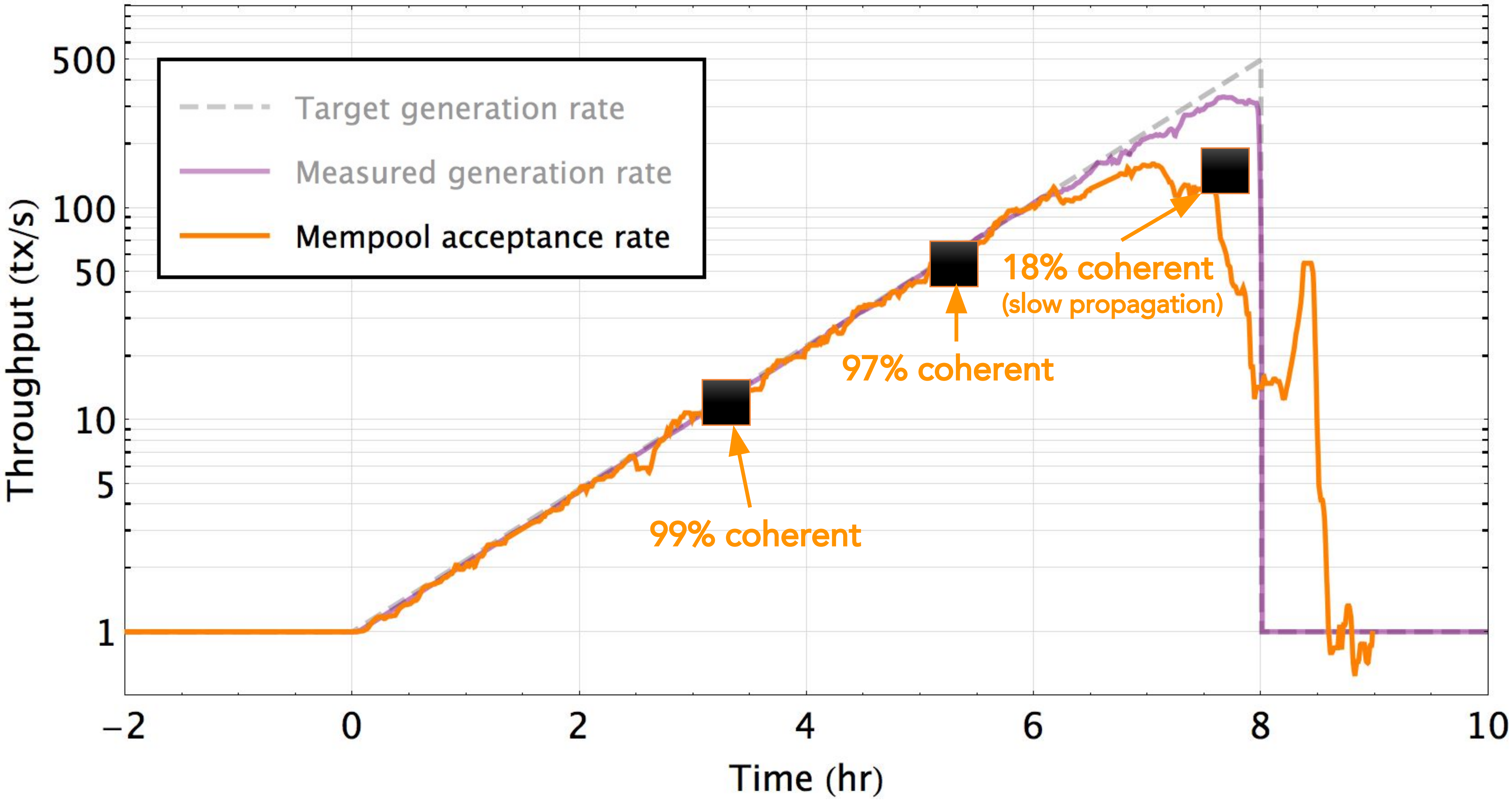


Target generation rate
Measured generation rate
Throughput (tx/s)
500
100
50
10
5
1
–2
0
2
4
6
8
10
Time (hr)

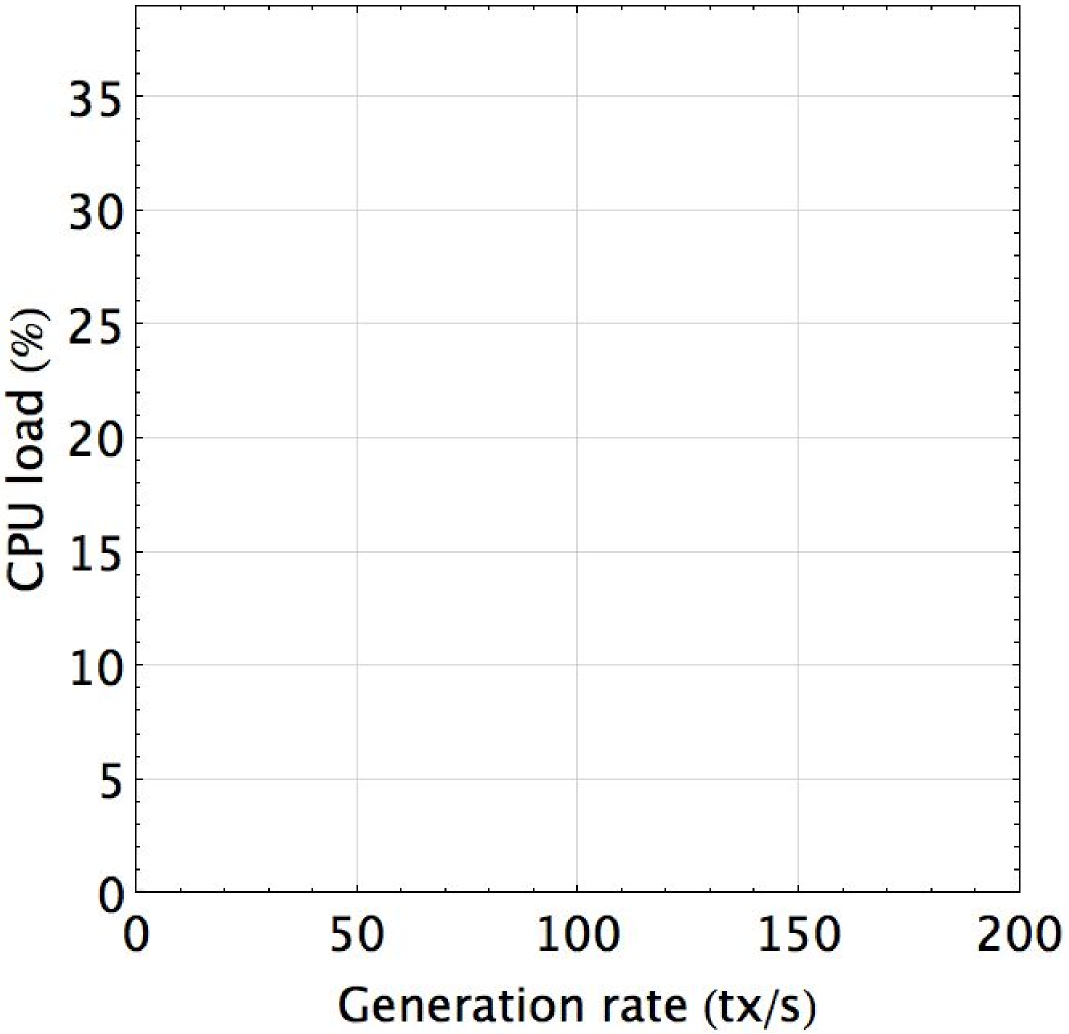
Ramp tests


Bottleneck #1: 100 tx/s
Mempool acceptance
cannot keep up
Target generation rate
Measured generation rate
Mempool acceptance rate
Mempool acceptance
keeps up
500
100
50
10
5
1
Throughput (tx/s)
–2
0
2
4
6
8
10
Time (hr)

Ramp tests

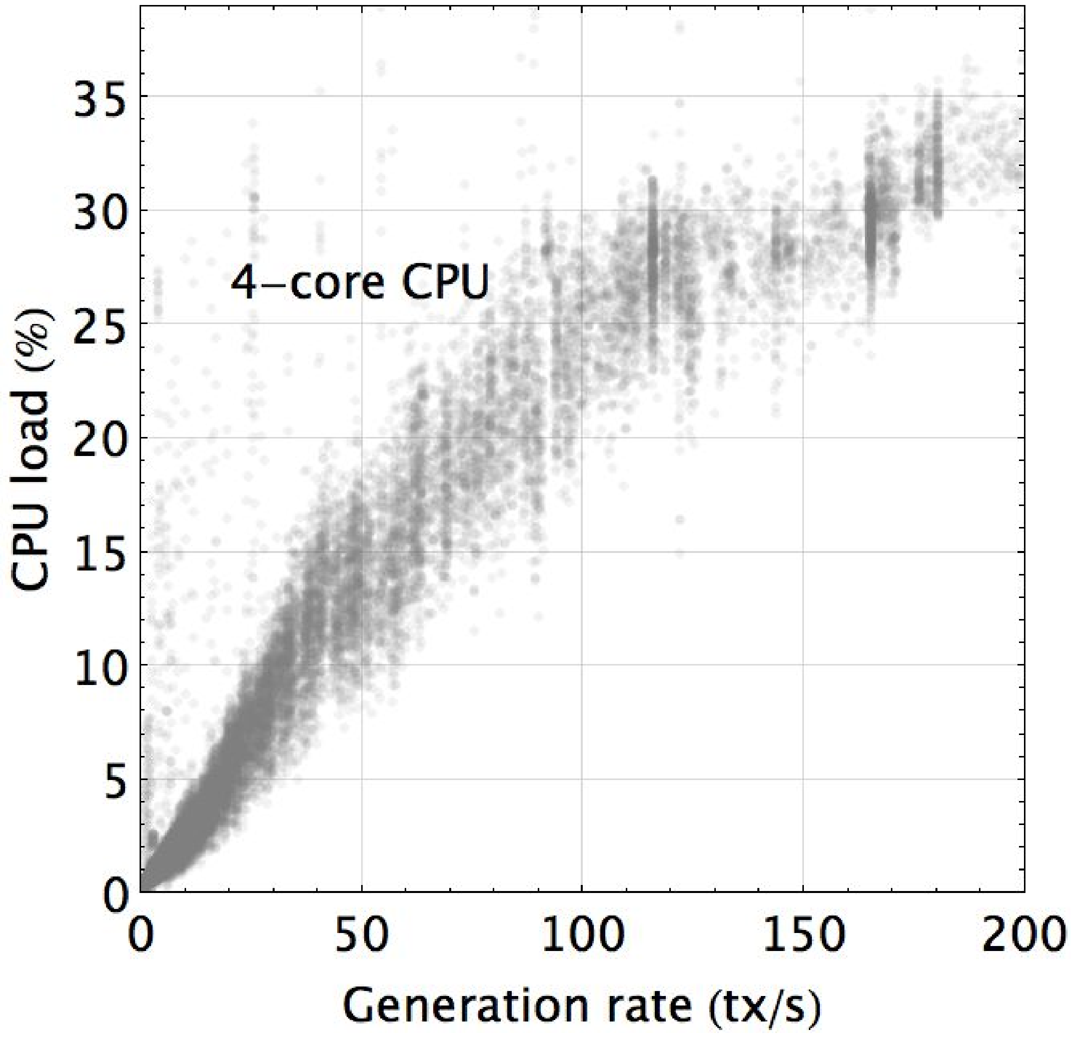

Target generation rate
Measured generation rate
Mempool acceptance rate
99% coherent
97% coherent
18% coherent
(slow propagation)
Throughput (tx/s)
500
100
50
10
5
1
–2
0
2
4
6
8
10
Time (hr)

What caused the bottleneck?



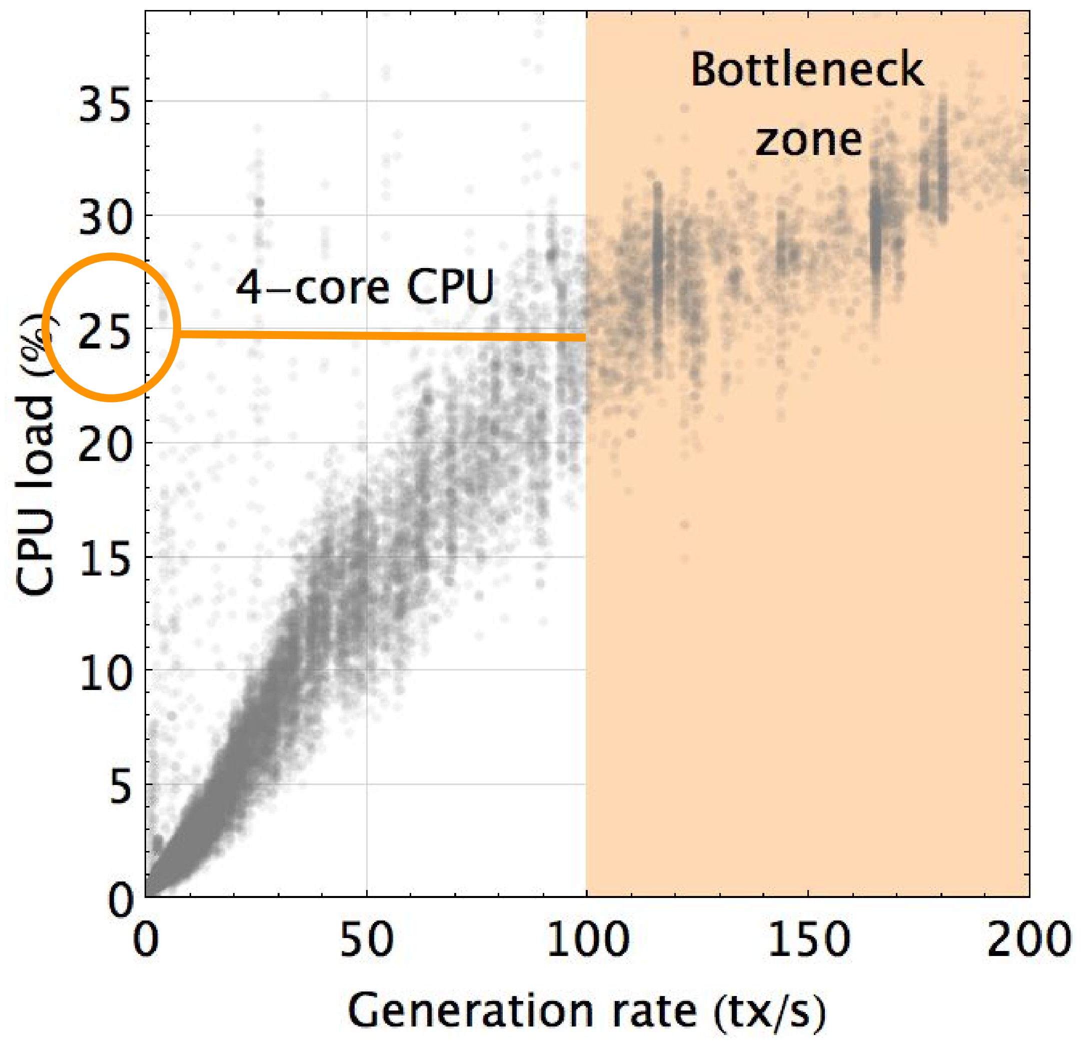
What caused the bottleneck?

- It wasn't the CPU
  - 25% of a 4-core machine at 100 tx/s



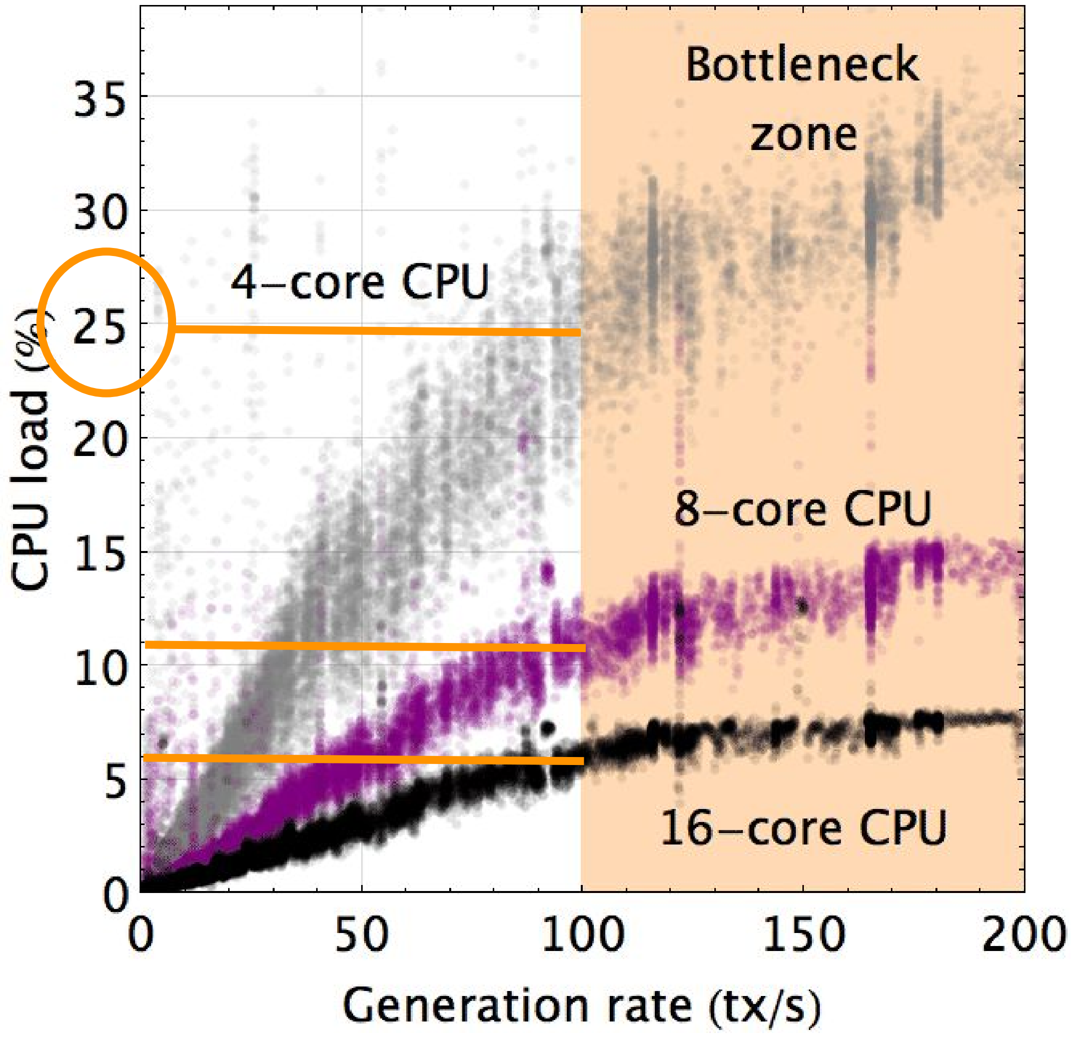
What caused the bottleneck?

- It wasn't the CPU
  - 25% of a 4-core machine at 100 tx/s



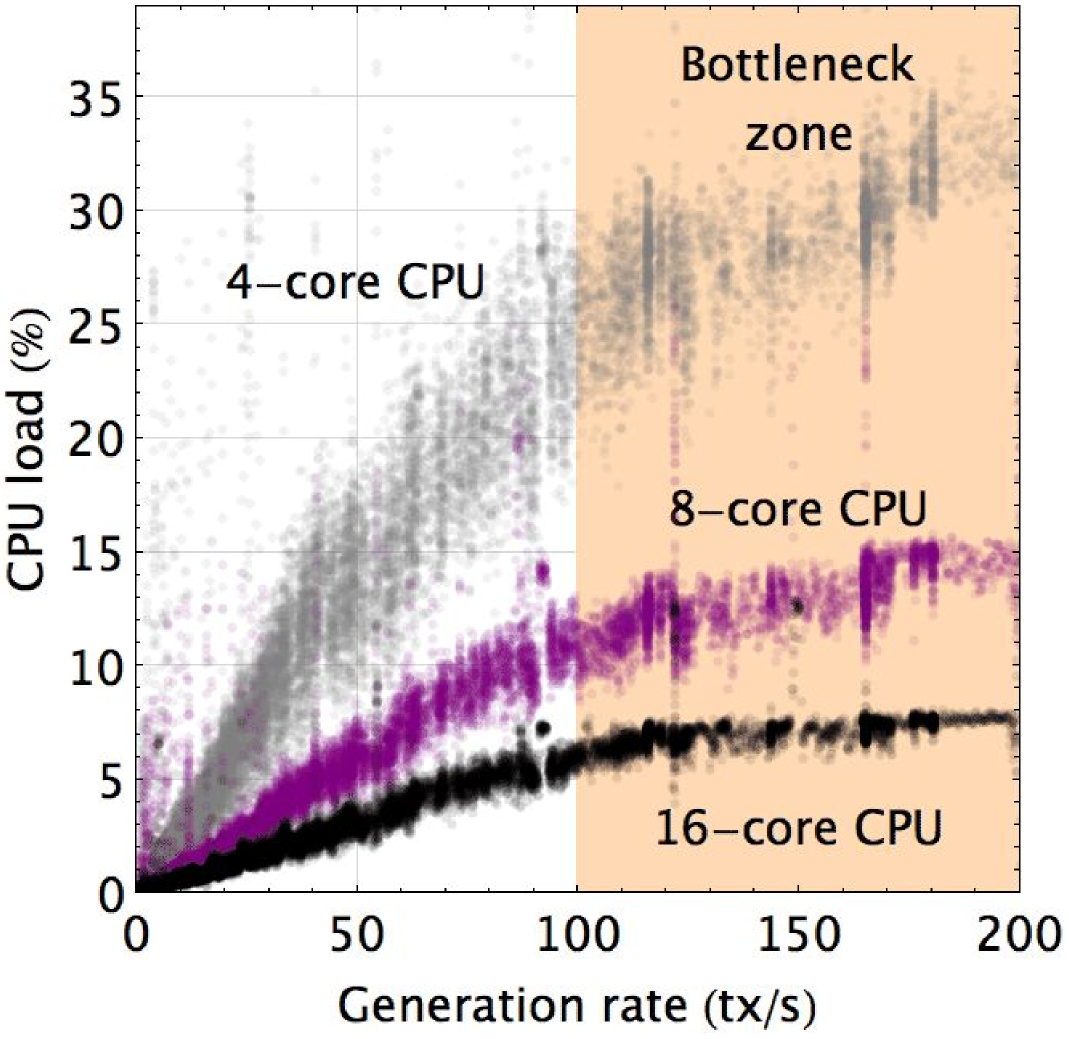
What caused the bottleneck?

- It wasn't the CPU
  - 25% of a 4-core machine at 100 tx/s

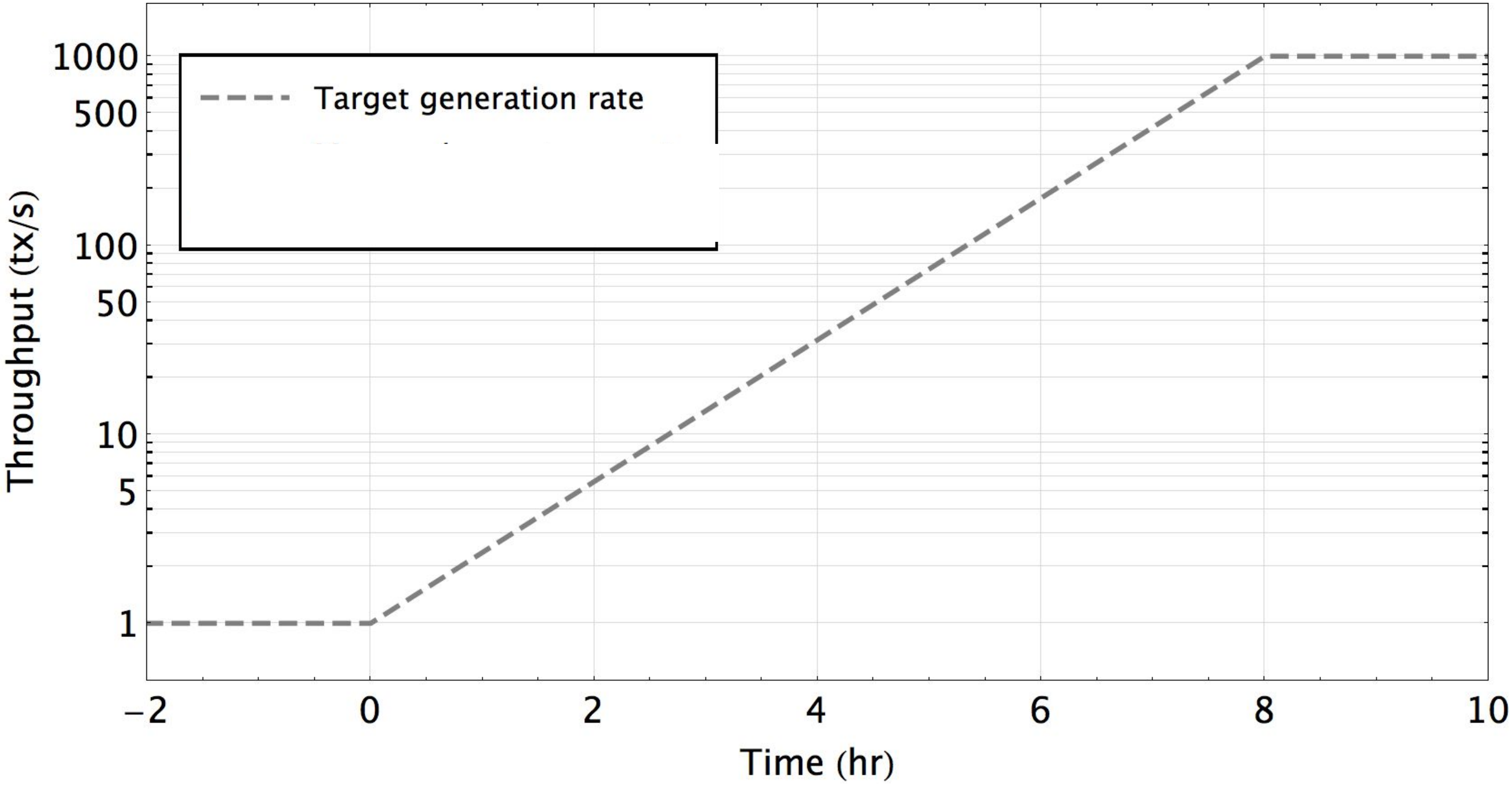


What caused the bottleneck?

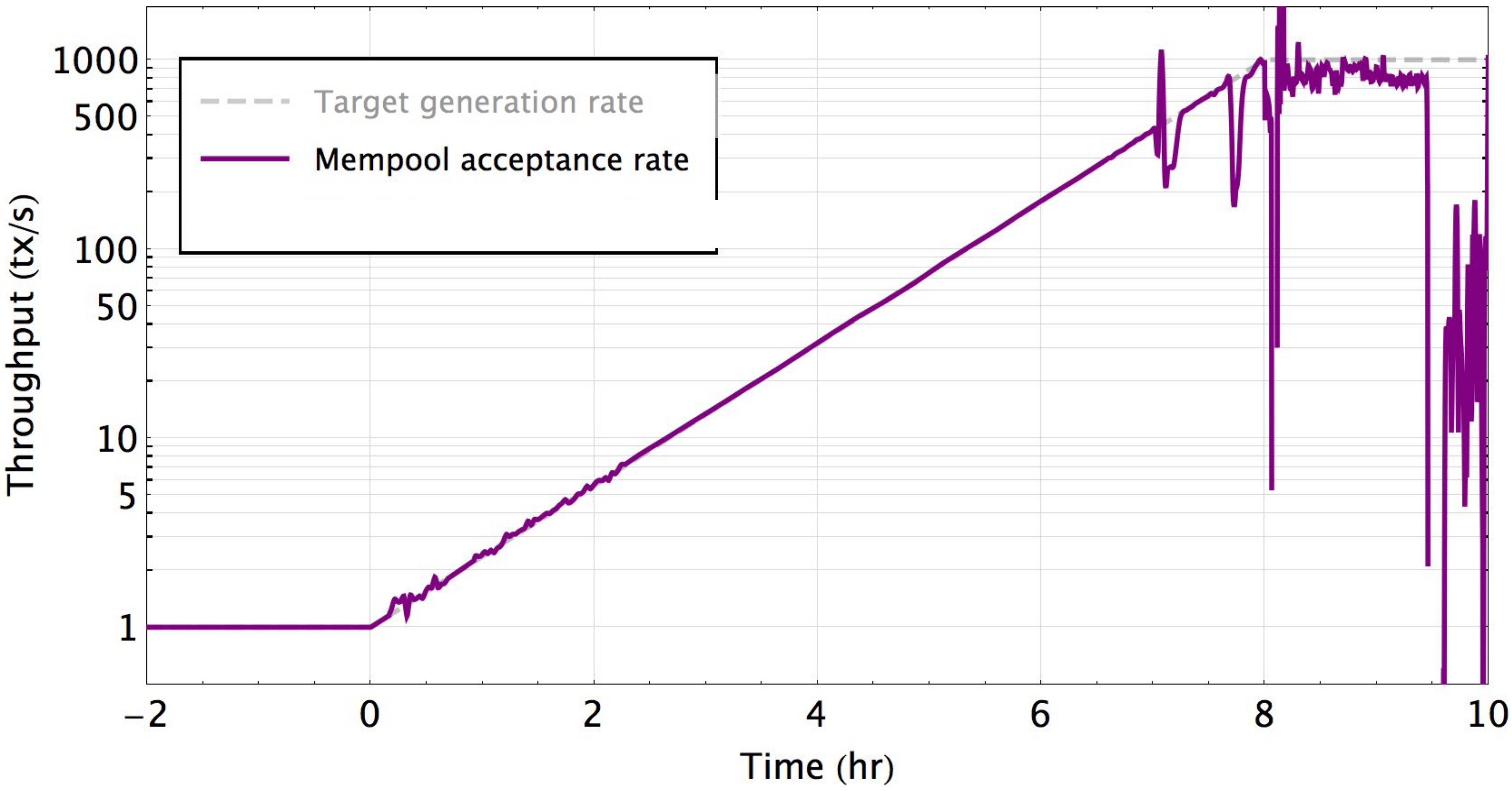
- It wasn't the CPU
  - 25% of a 4-core machine at 100 tx/s
- It <u>was</u> the single-threaded mempool acceptance code path
- Andrew Stone parallelized mempool acceptance
  - Now can achieve over 1,000 tx/sec sustained
  - Bursts over 10,000 tx/s on strongest nodes



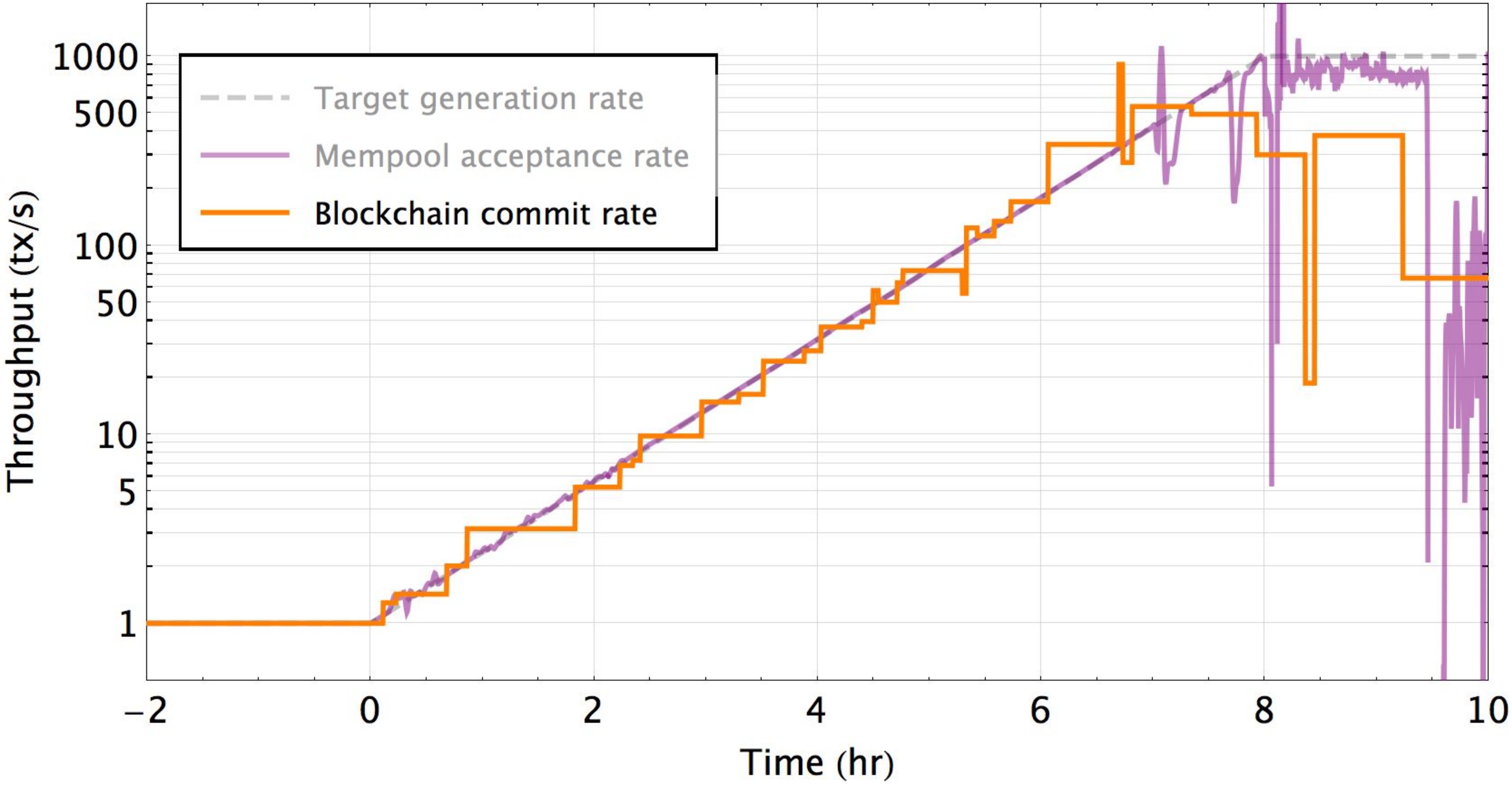
Ramp tests with bottleneck removed



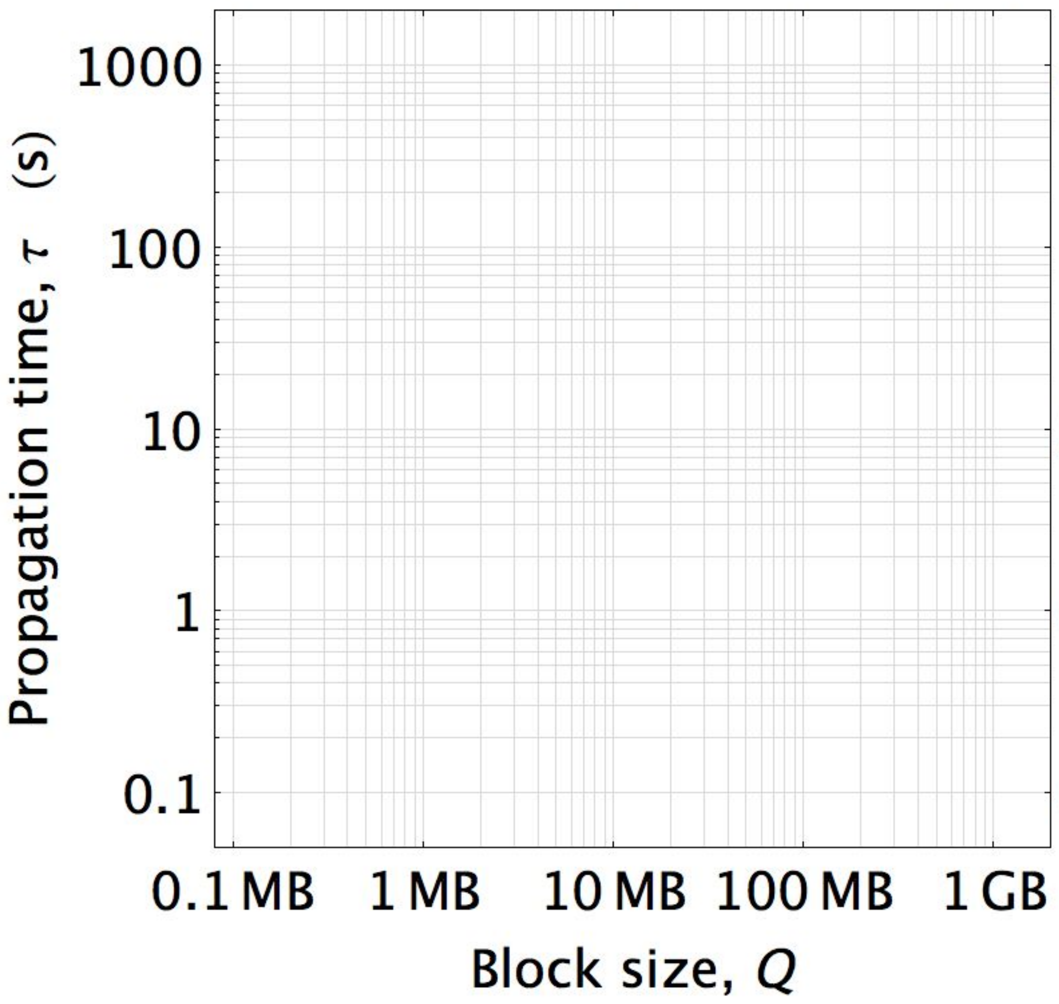
Ramp tests with bottleneck removed



Ramp tests with bottleneck removed



Xthin block propagation



Xthin block propagation

Linear model:

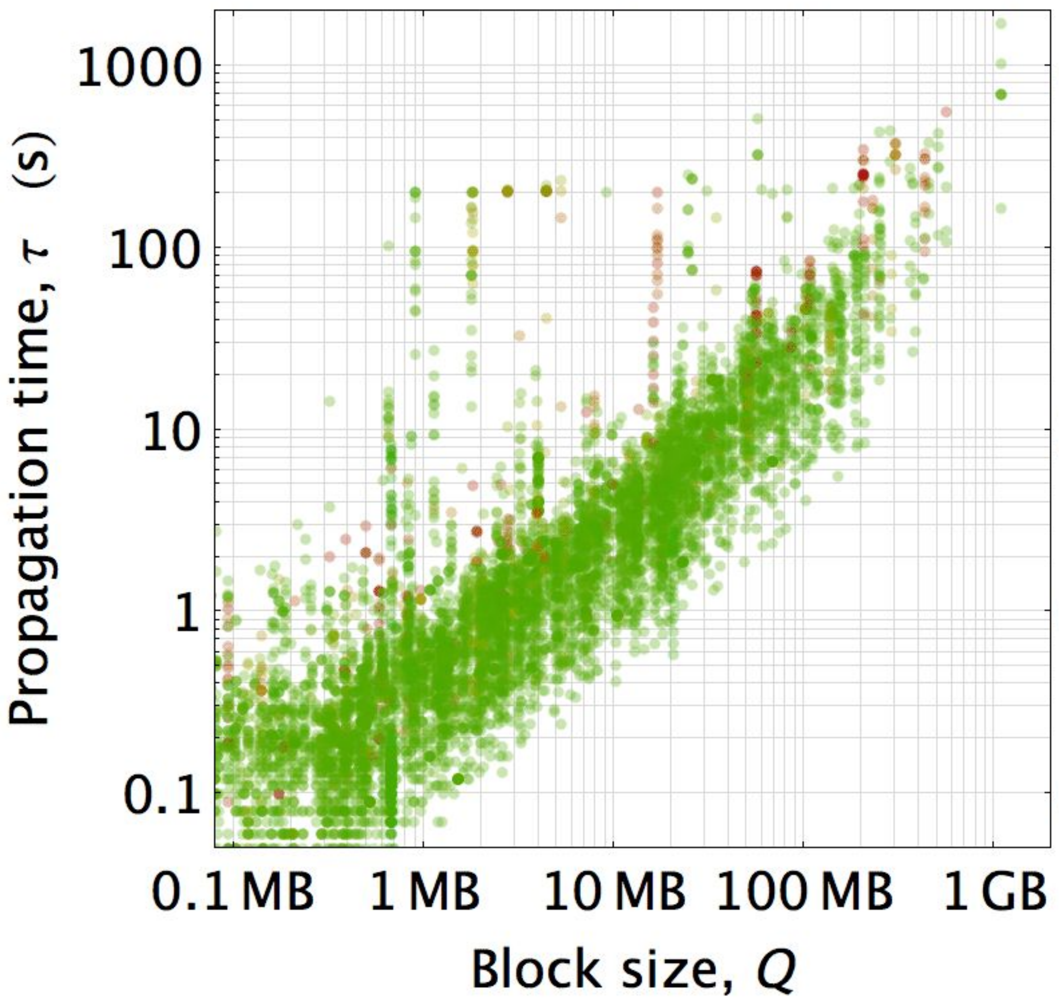
Propagation time → τ; Block size → Q

$$\tau = \tau_0 + zQ$$

Empty-block time → τ_0; Propagation impedance → z

Least-squares best fit:

$$\tau_0 = 0.2\ \mathrm{s}$$
$$z = 0.6\ \mathrm{s/MB}$$



Xthin block propagation

Linear model:

Propagation time — Block size

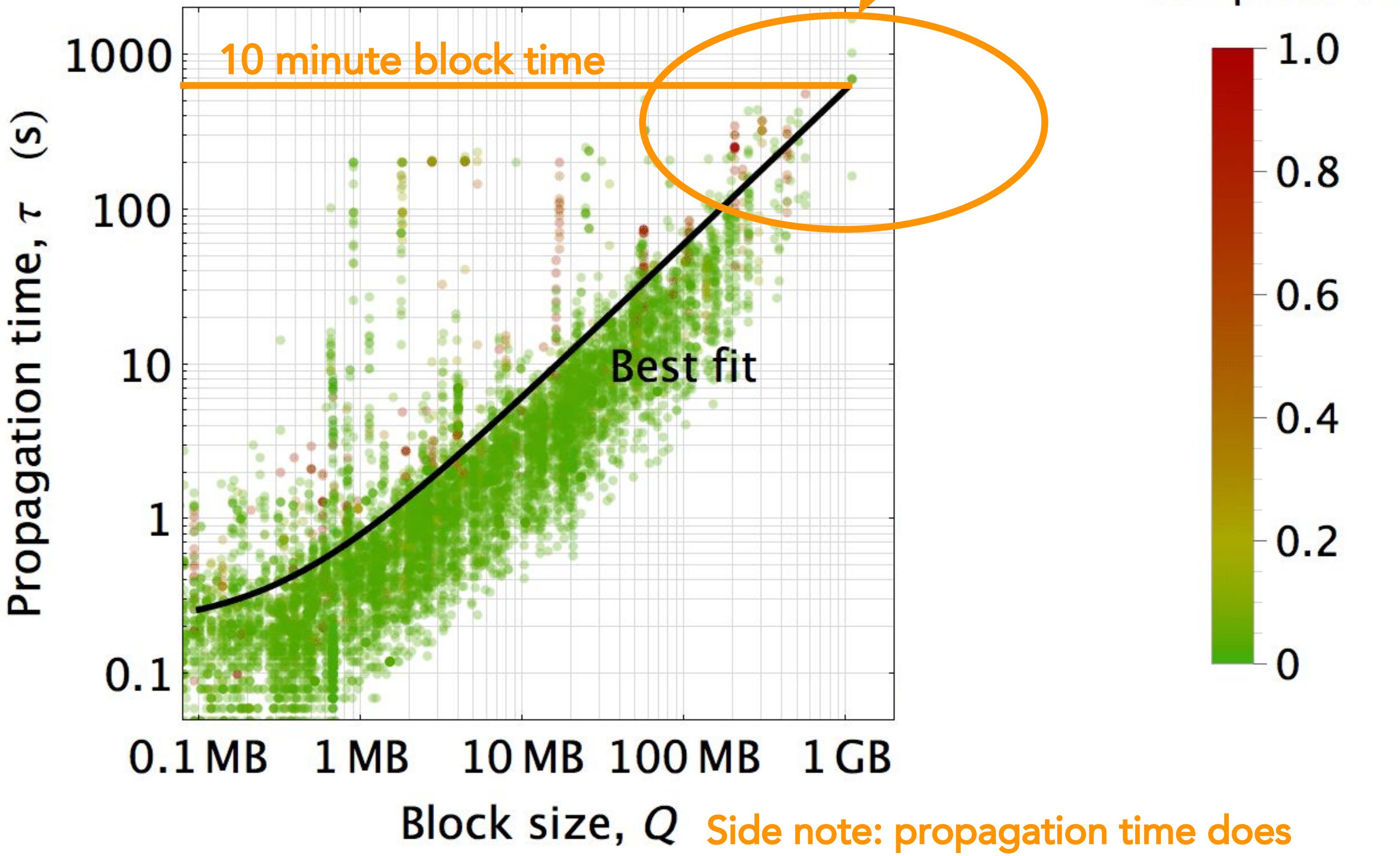
$$\tau = \tau_0 + zQ$$

Empty-block time — Propagation impedance

Least-squares best fit:

$$\tau_0 = 0.2\ \mathrm{s}$$
$$z = 0.6\ \mathrm{s/MB}$$

Bottleneck: propagation time commensurate with 10 min block time



Side note: propagation time does not depend strongly on network BW

Regressions, interpolations & extrapolations

	Regression coefficient	100 tx/sec (mempool bottleneck)	2000 tx/sec (Visa level)	50,000 tx/sec (global adoption)
CPU	0.01 cores / (tx/sec)	1 core	20 cores	500 cores
Network	0.03 Mbps / (tx/sec)	3 Mbps	60 Mbps	1.5 Gbps
Memory				
Disk IO				

Regressions, interpolations & extrapolations

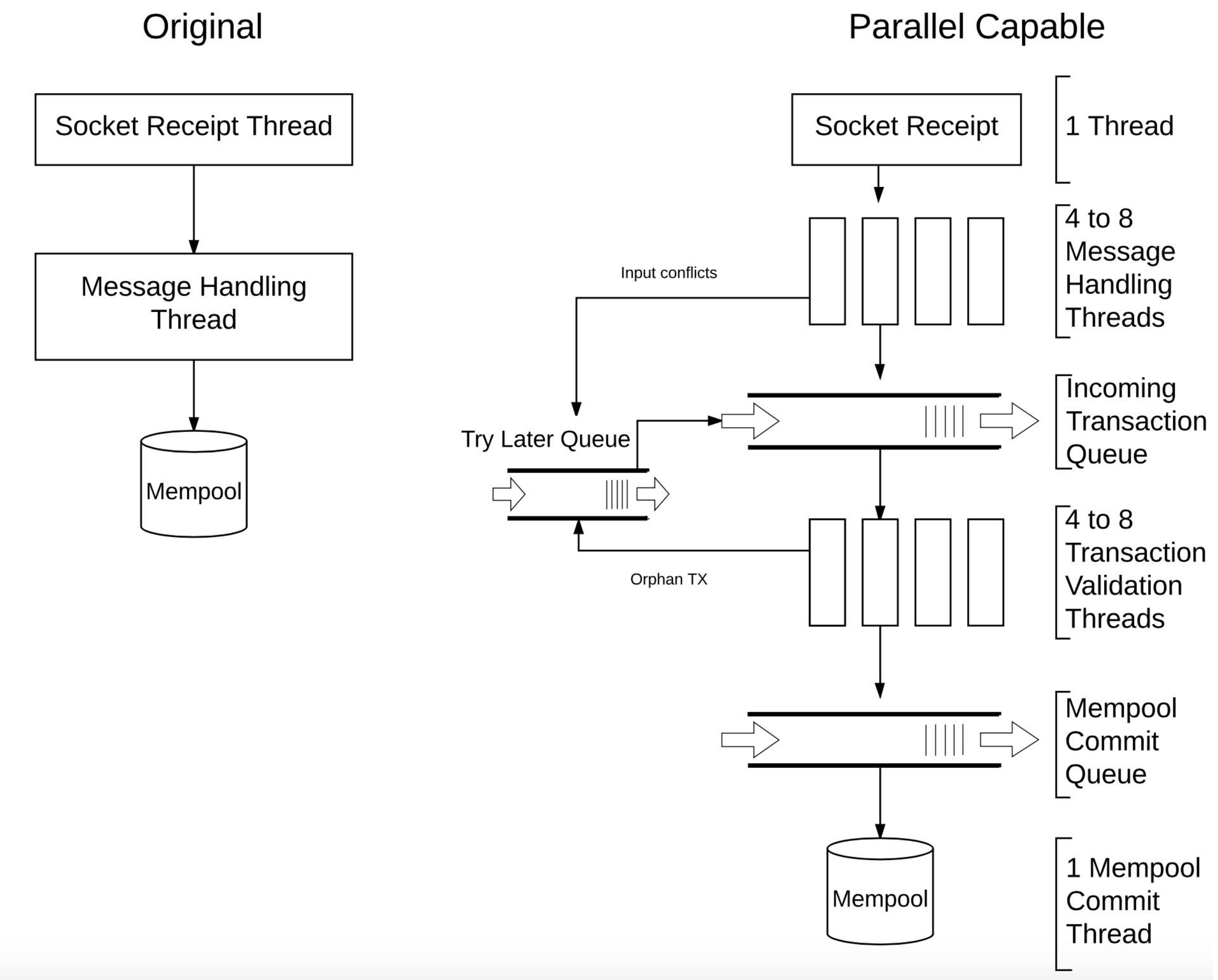
	Regression coefficient	100 tx/sec (mempool bottleneck)	2000 tx/sec (Visa level)	50,000 tx/sec (global adoption)
CPU	0.01 cores / (tx/sec)	1 core	20 cores	500 cores
Network	0.03 Mbps / (tx/sec)	3 Mbps	60 Mbps	1.5 Gbps
Memory				
Disk IO				

Bottlenecks were neither the protocol nor the infrastructure.
The bottlenecks were in the implementation of the protocol.

TBD in Experiment #2: UTXO stress test

My hunch: we can achieve Visa level with 4-core/16GB machines with better implementations.

Transaction Processing Architecture



Locking Strategy

Reduce use of cs_main

- Transition to fine-grained locking

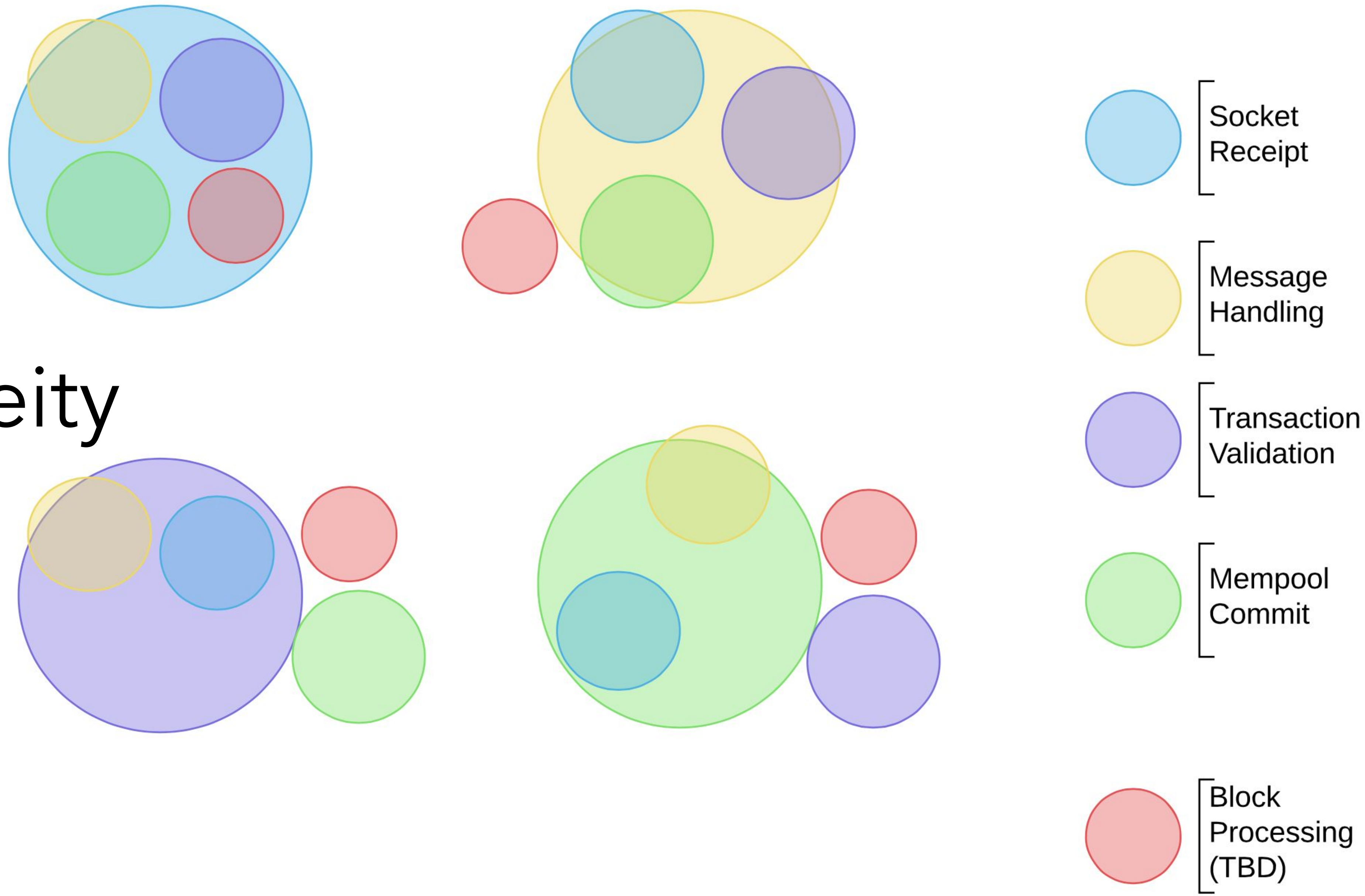
Use shared mutexes

- Shared mutexes allow simultaneous readers and an exclusive writer.

- Most boost and std containers have the same access semantics.

- Apply to all major state:
  - mempool,
  - UTXO set,
  - chain state,
  - orphan pool,
  - recent rejects,

Simultaneity



Optimizations

Fast Bloom Filter

Validate transactions once

Stop block re-serialization just to determine size

AlreadyHave() locks and touches everything

Fast Coin Selection

N^2 txn processing in block
(ConnectTip->SyncWithWallets)

Reduce cs_main scope
(orphan list, versionbitscache, recentRejects, AlreadyHave(), chainActive tip)

Move locking out of tight loops

Do not format logs that won't be issued

std::atomic
(chainActive.Tip())

Use shared locks
(mempool, orphan cache, recent rejects, utxo)

Sharded request manager

remove extraneous sha256 hashing
(tx trickle, save block and tx id)

Message processing chunking

Don't hold locks across disk accesses and logging

Scaling Fixes

Stop Copying Blocks By Value

Fix hang when block is larger than max block file size

Increase max buffer sizes

UTXO "Coin" returned by reference, lock released

Do not rerequest a block if it is being processed

Fast Bloom Filter

Is it likely that I've seen this data before?

Bloom Filter

Hn = Execute N hash functions over data

INSERT: Set every bit Hn

CHECK: If every bit Hn is set return TRUE

Observations:

Hashing is extremely slow

We can't get more random than the SHA256 cryptographic hash algorithm used to create transaction and block id's

Why are we hashing the hash?

Fast Filter

Hn = select arbitrary subsets of TX hash

Use power of 2 filter buffer size for fast math

INSERT: Set every bit Hn

CHECK: If every bit Hn is set return TRUE

CHECK_INSERT: doing it together halves number of memory accesses

Note:

Each node chooses random "arbitrary subsets of the TX hash" so attackers cannot reliably fabricate collisions

Thank you!

Funding provided by

bitcoinunlimited nChain

Code at https://github.com/gandrewstone/BitcoinUnlimited
"giga_perf" branch

GTI contact person
peter.rizun@gmail.com

The Next 25 Years for Bitcoin: A Payment Network for Planet Earth