Experimental Analysis of the Performance and Scalability of Network Time Security for the Network Time Protocol

Griffin Leclerc

University of New Hampshire
InterOperability Laboratory
Department of Computer Science

gleclerc@iol.unh.edu

Radim Bartos

University of New Hampshire
Department of Computer Science

rbartos@unh.edu

This research has been supported in part by a gift from the Internet Society
to assist in the establishment of a Community Network Time Security Lab at the
UNH IOL

October 5th, 2022

Vulnerabilities of NTP

Spoofing - An unqualified attacker acts as a timing master to distribute false timing information

Man in the Middle (MITM) - Modification of in flight NTP requests to inject incorrect timing information

Replay - An attacker modifies an replays a previous NTP response to convey incorrect timing information

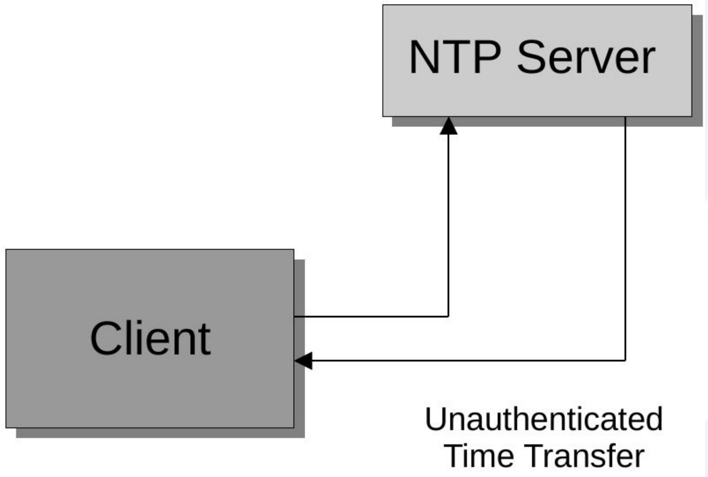
Objectives of NTS

Address the vulnerabilities of NTP through identity verification and authentication

While maintaining a high level of scalability and performance

Need for Message Security

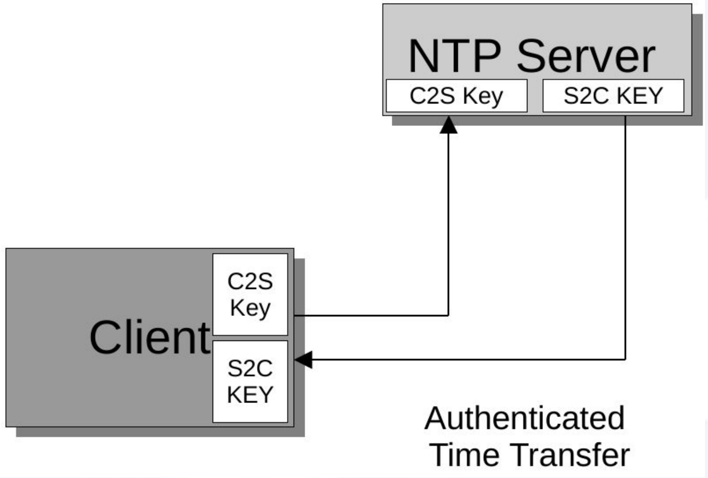
To ensure authenticity of messages, encryption keys must be introduced.



Need for Key Distribution

To ensure authenticity of messages, encryption keys must be introduced.

These keys must be securely and dynamically distributed among all nodes.

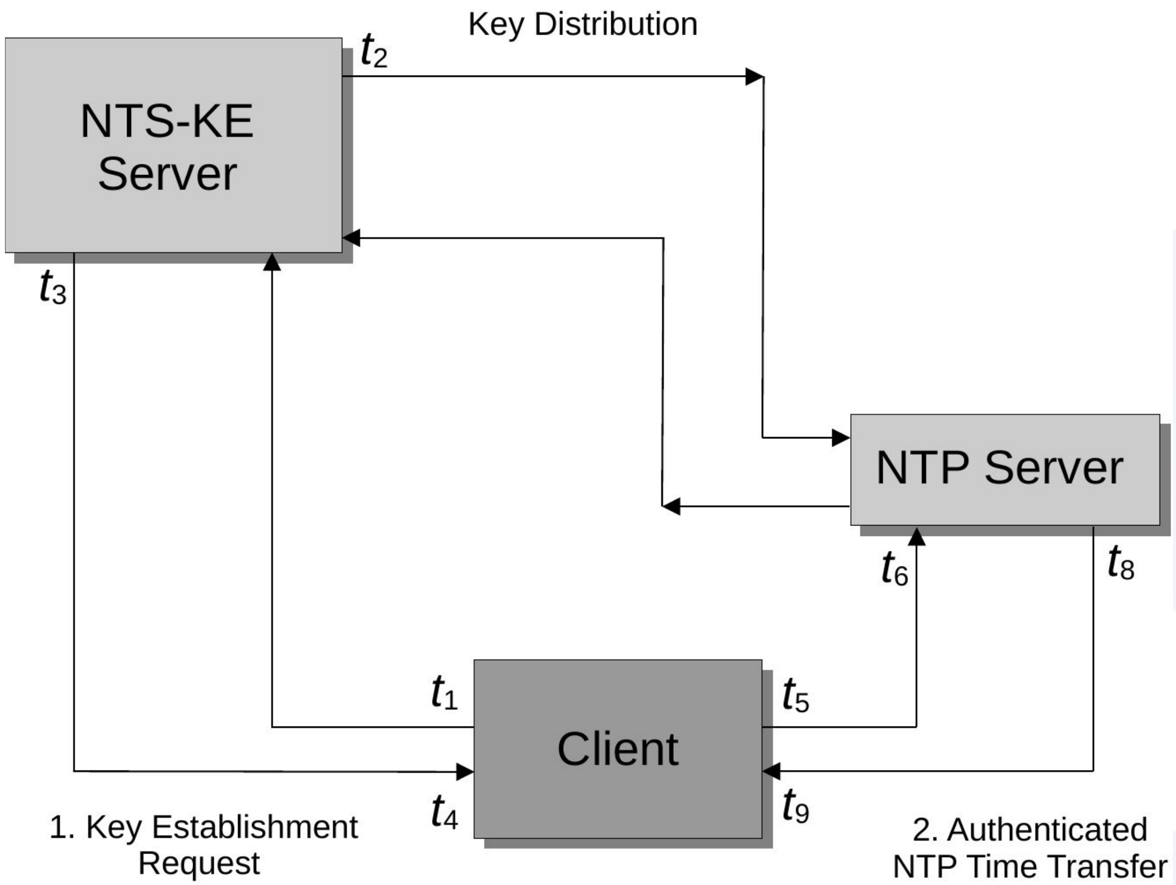


Need for Scalability

To ensure authenticity of messages, encryption keys must be introduced.

These keys must be securely and dynamically distributed among all nodes.

Multiple NTP servers should be available for scalability.



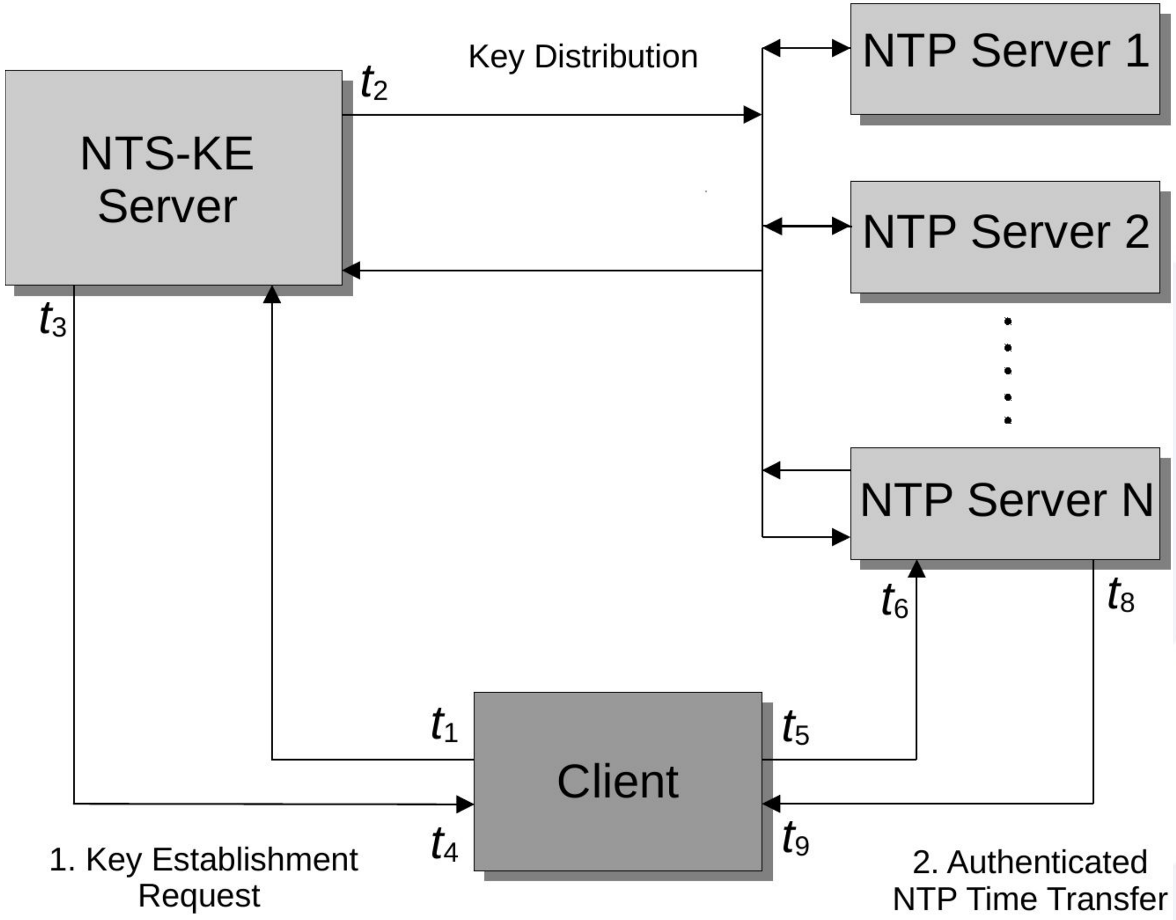
Need for Stateless Operation

To ensure authenticity of messages, encryption keys must be introduced.

These keys must be securely and dynamically distributed among all nodes.

Multiple NTP servers should be available for scalability.

NTP Servers should not maintain state.



Stateless NTS Operation

NTP Servers should not maintain a local key pair for each NTP client

- Data transfer overhead
- Keys must be rotated

A secure cookie is defined which allows clients to maintain their own local state

NTS-KE Cookie Format

Servers generate a secret master AEAD key K and unique value I to identify K

Servers form a plaintext, P containing:

- The AEAD algorithm negotiated during NTS-KE
- The S2C key
- The C2S key

Encrypting P with a nonce N under K results in the ciphertext C

The cookie consists of (I, N, C)

From RFC8915

NTS Authenticator Extension Field

A: The associated data, consisting of the NTP packet beginning from the start of the NTP header and ending at the end of the last extension field

P: any additional NTP extension fields to be encrypted

N: The nonce required by the negotiated AEAD algorithm

K: either the C2S or S2C encryption key, depending on message direction

The Encrypted Extension field for NTPv4 consists of (***A***, ***P***, ***N***) encrypted by ***K***

From RFC8915

Network Time Security Mechanisms

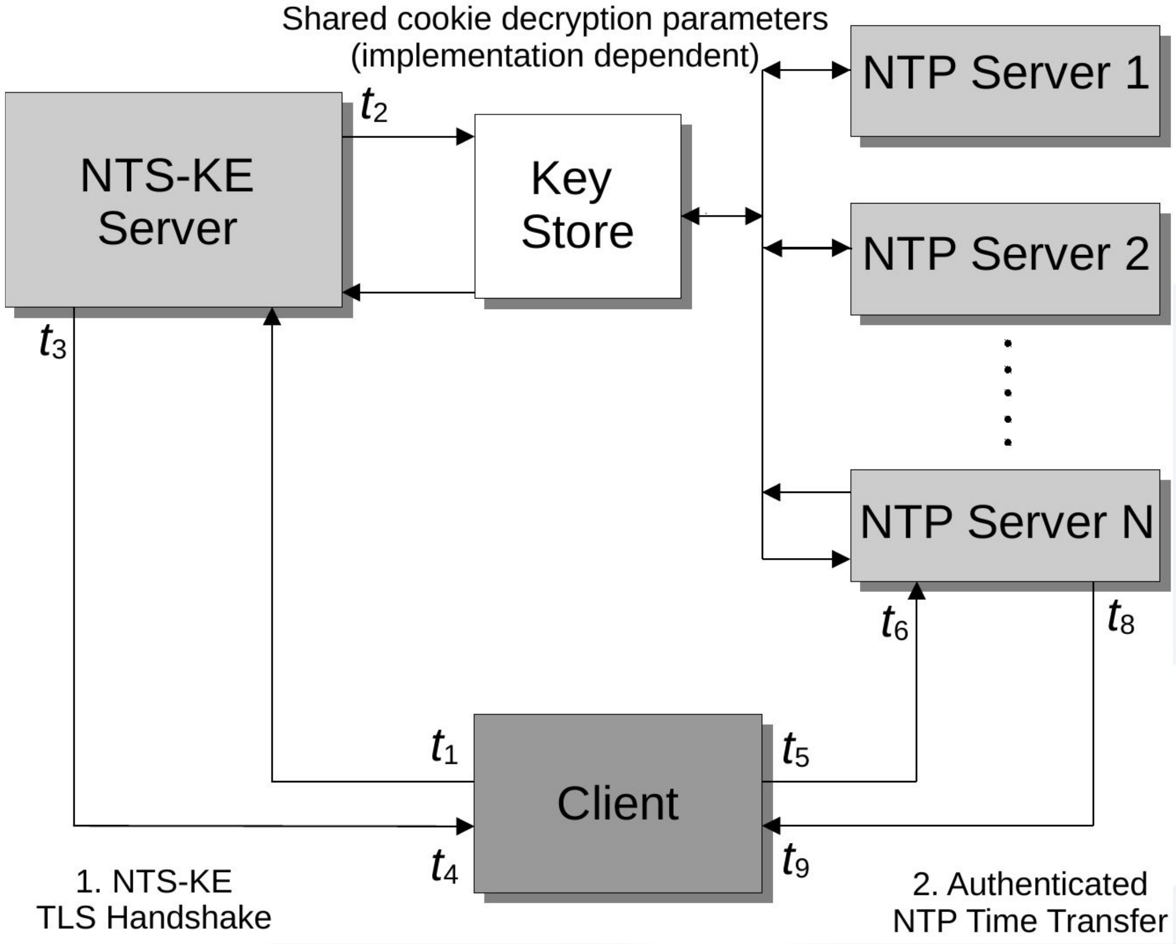
Two protocols are defined:

<u>NTS-KE</u>:

Clients obtain an encrypted cookie from the NTS-KE server via TLS, t_1 to t_4

<u>Extension Fields for NTPv4</u>:

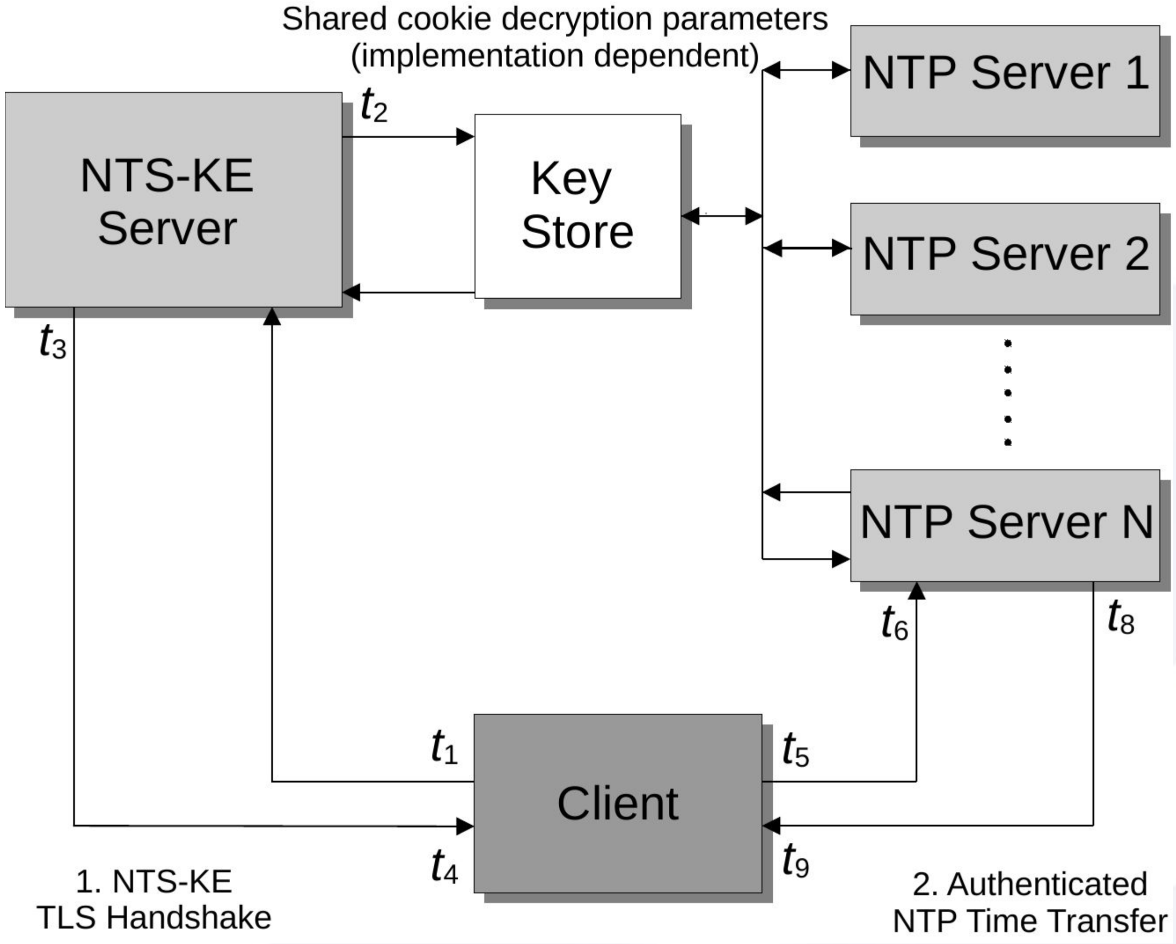
Secure NTPv4 with shared encryption parameters from NTS-KE, t_5 to t_9



An adaptation of Figure 1 from RFC8915

Network Time Security Guarantees

- The NTP server's ability to decrypt the cookie proves it is a trusted member of the same NTP domain
- Decrypting the extension field from the NTP server verifies the integrity of the packet
- A nonce in the extension field prevents replay



An adaptation of Figure 1 from RFC8915

Performance and Scalability

This paper involves two studies of NTS:

Performance - Quantify difference in time transfer completion when compared to base NTP

Scalability - Observe protocol performance as network load increases

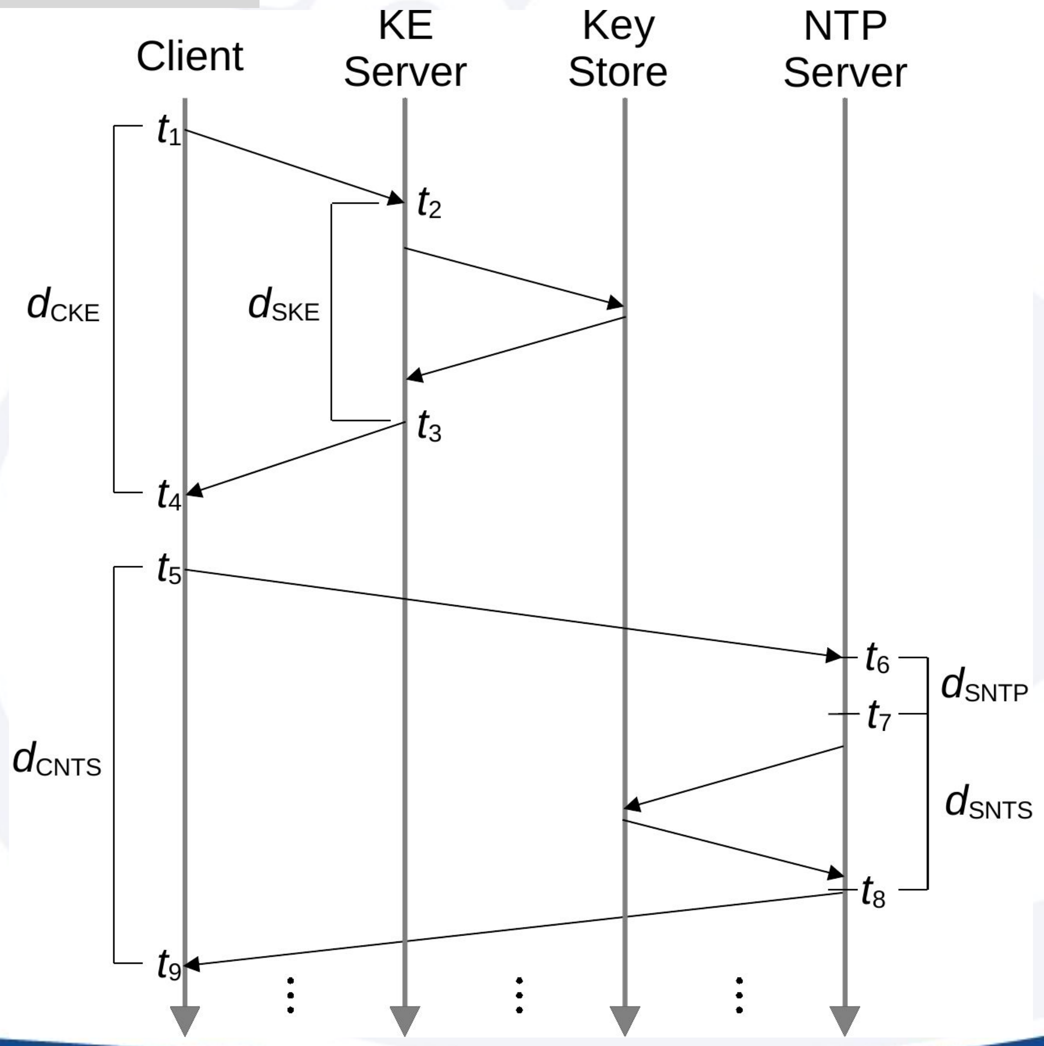
Client Measurement Definitions

$d_{CKE} = t_4 - t_1$

Time for a client to receive the initial encrypted cookie

$d_{CNTS} = t_9 - t_5$

Time for a client to conduct authenticated time transfer



Server Measurement Definitions

$d_{SKE} = t_3 - t_2$

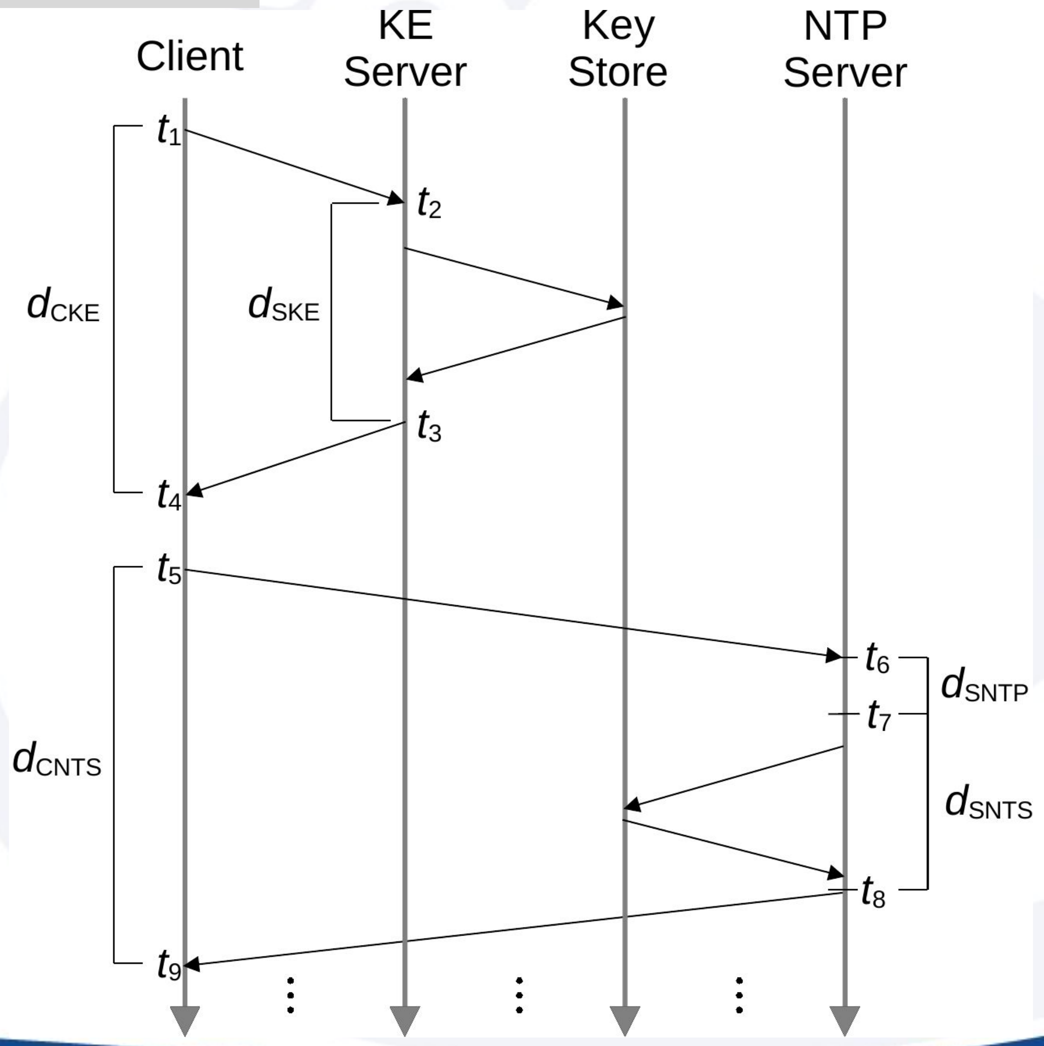
Time for a KE server to create an encrypted cookie

$d_{SNTP} = t_7 - t_6$

Time for an NTP server to create an NTPv4 header

$d_{SNTS} = t_8 - t_7$

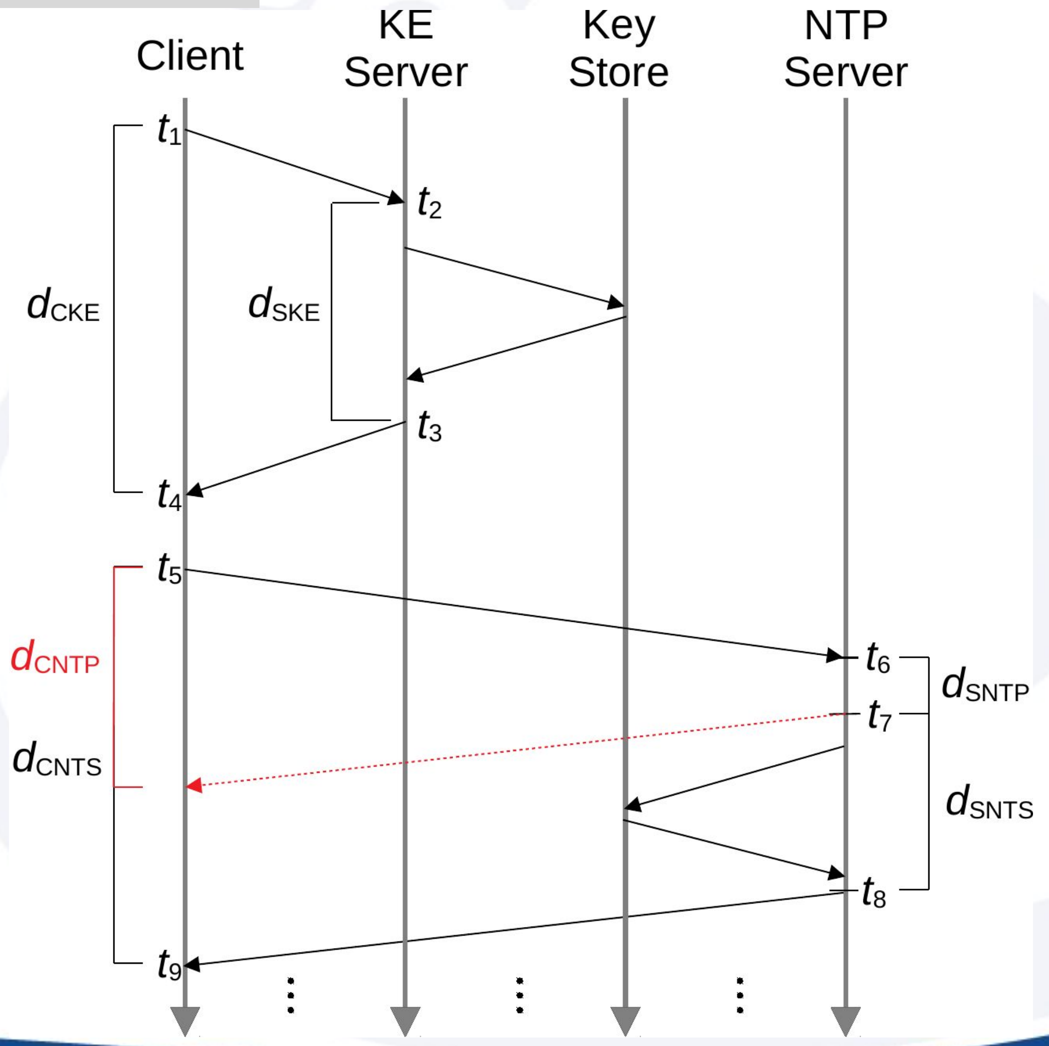
Time for an NTP server to process a cookie and authenticate an NTPv4 message



Other Measurement Definitions

$$d_{\mathrm{CNTP}} = d_{\mathrm{CNTS}} - d_{\mathrm{SNTS}}$$
$$= (t_9 - t_5) - (t_8 - t_7)$$

Calculated approximation of unauthenticated time transfer



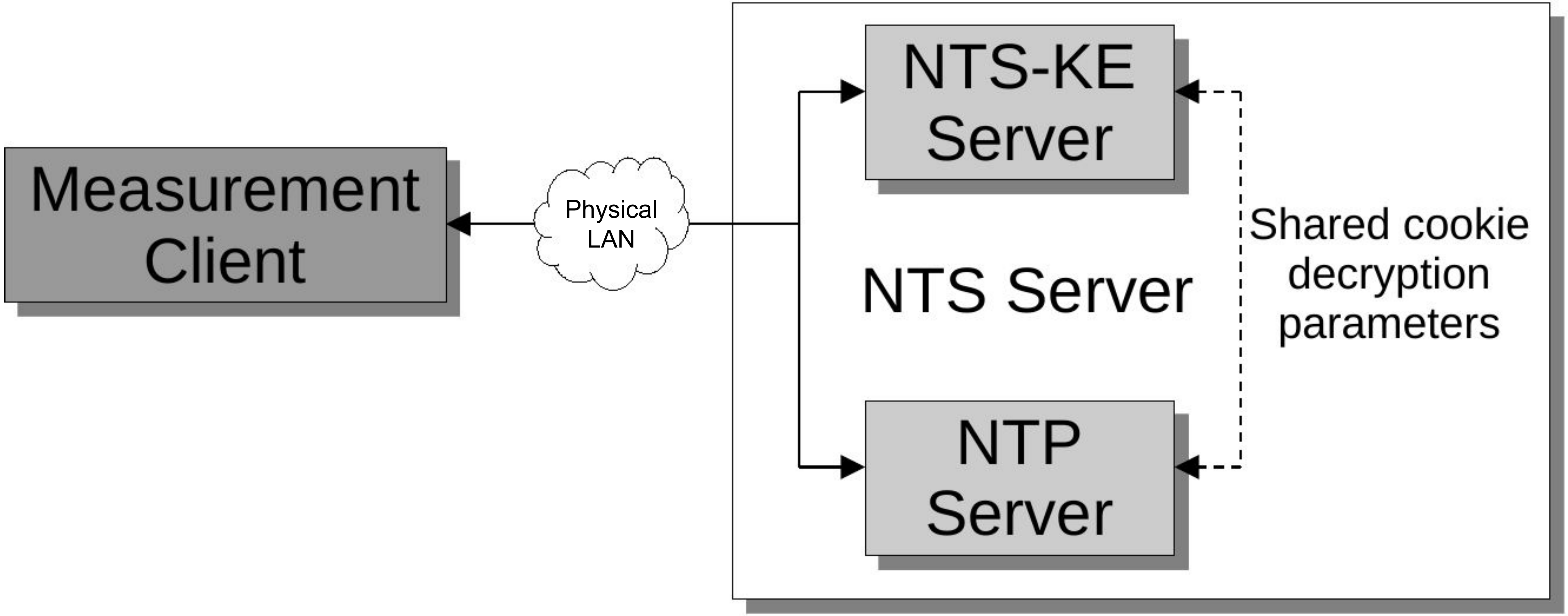
Performance Study

Quantify any time transfer performance impact introduced by NTS mechanisms

Isolate NTS operation from NTP

Augment Cloudflare’s open source NTS implementation with Rust standard library functions

Performance Experiment Topology



Performance Environment Details

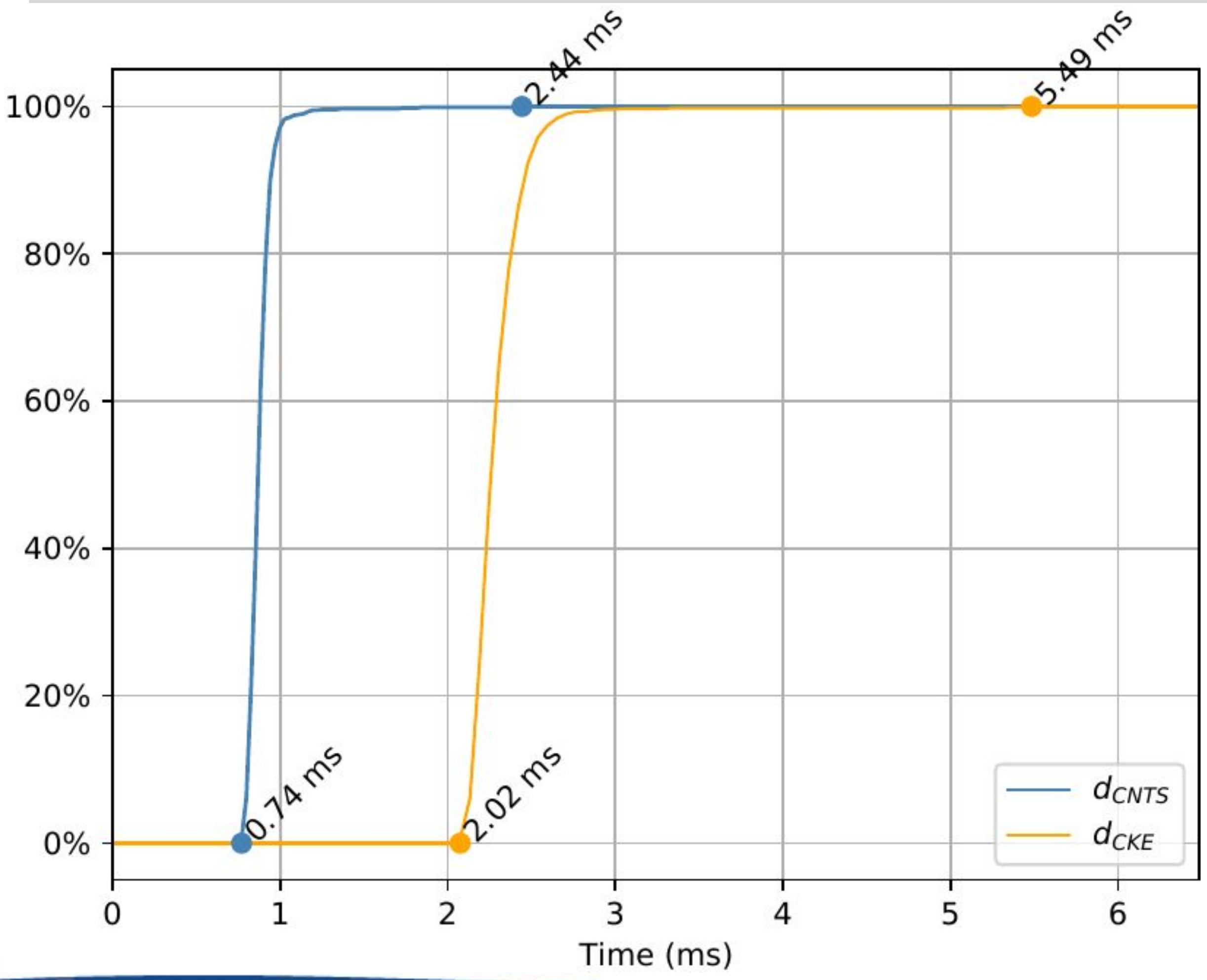
Client - Virtual machine with eight cores and 16 GB of memory

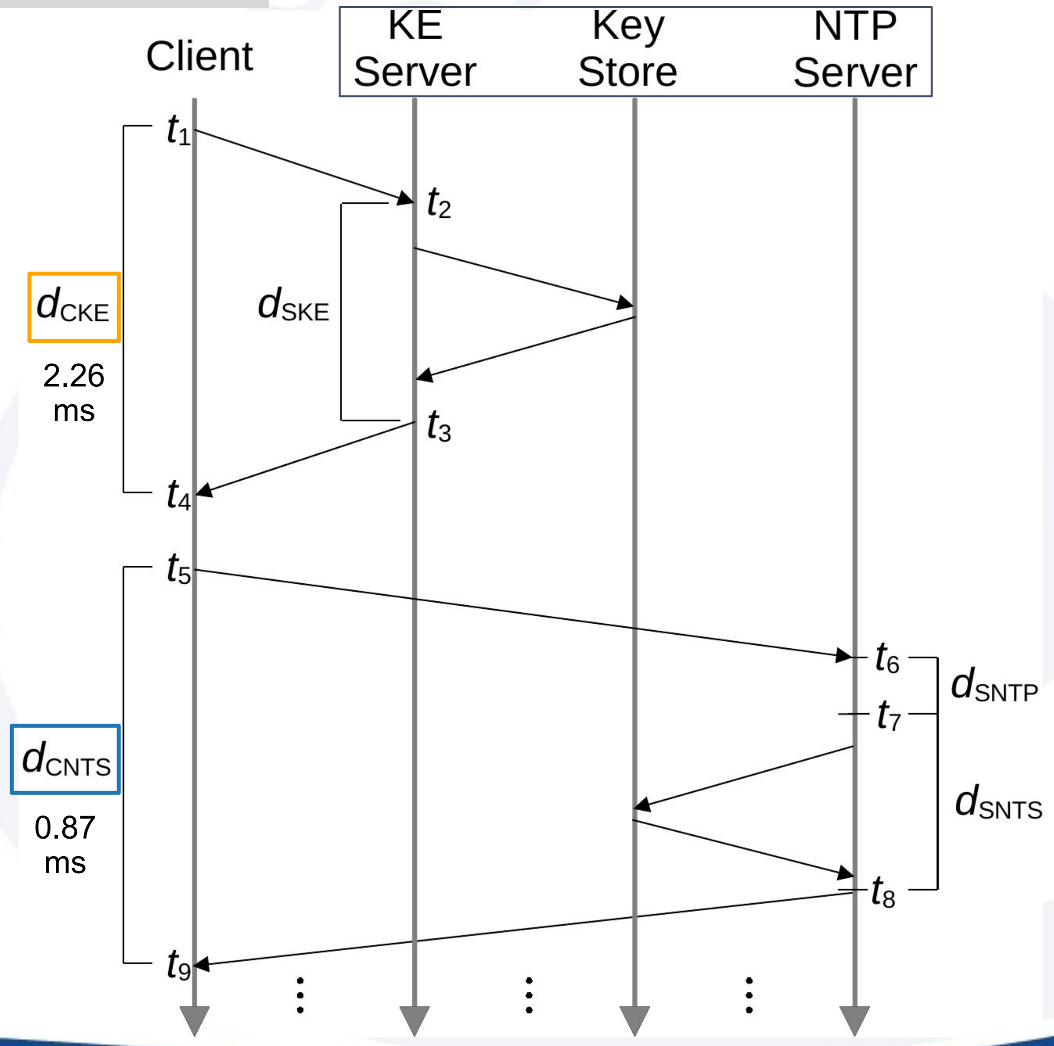
- Transmitted one NTP-KE request and one NTP request per second for 1000 seconds

Server - Physical machine with a 4-core 3.3 GHz Intel i5-2500k and 24 GB of memory

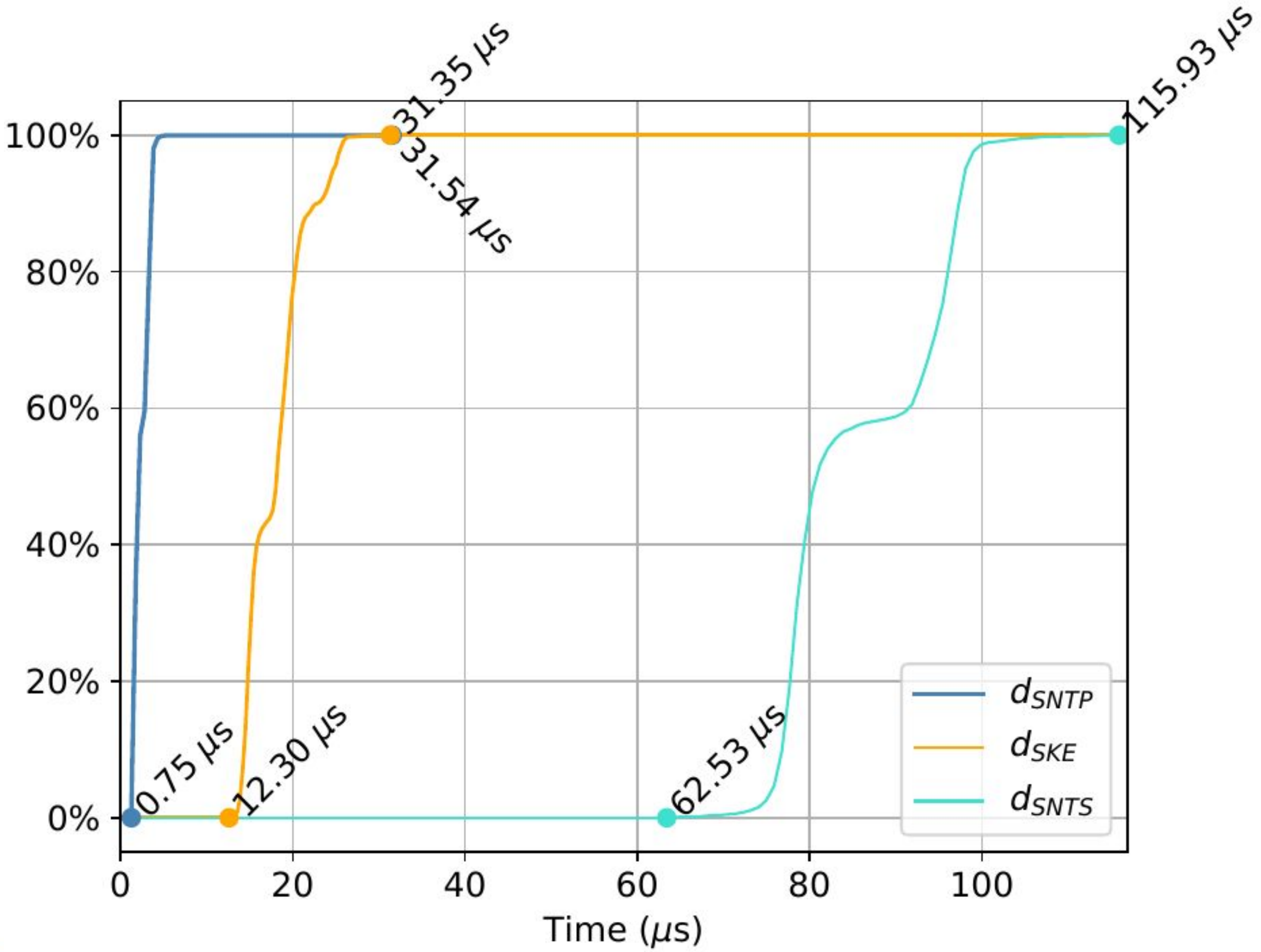
- Hosted both the NTS-KE and NTP server

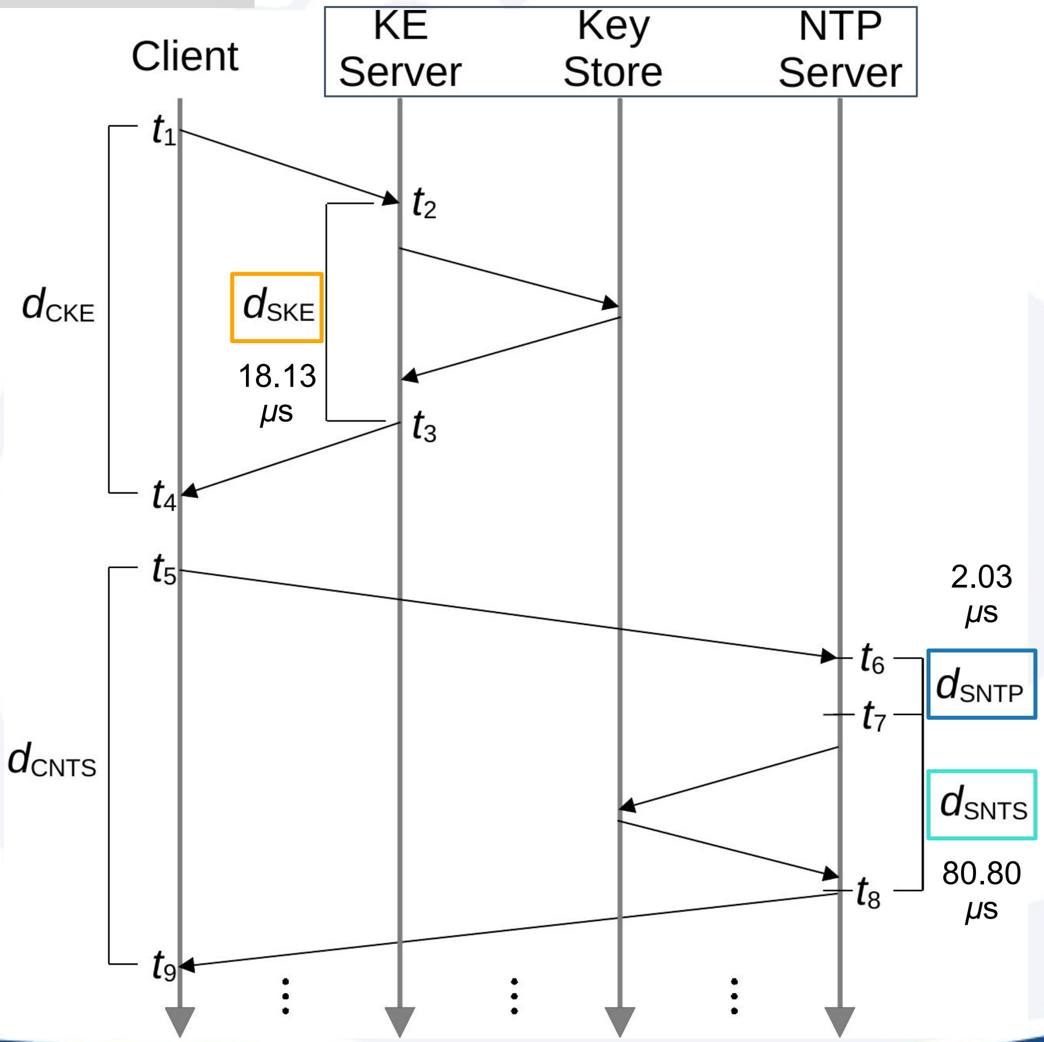
Client Measurement CDFs





Server Measurement CDFs





Performance Results

Client:

- NTS-KE adds an overhead of 2.26 ms
- Unauthenticated NTPv4 (d_{CNTP}) takes 0.79 ms
- Authenticated NTPv4 (d_{CNTS}) takes 0.87 ms
  - A 9.73% increase in time required to conduct time transfer

Server:

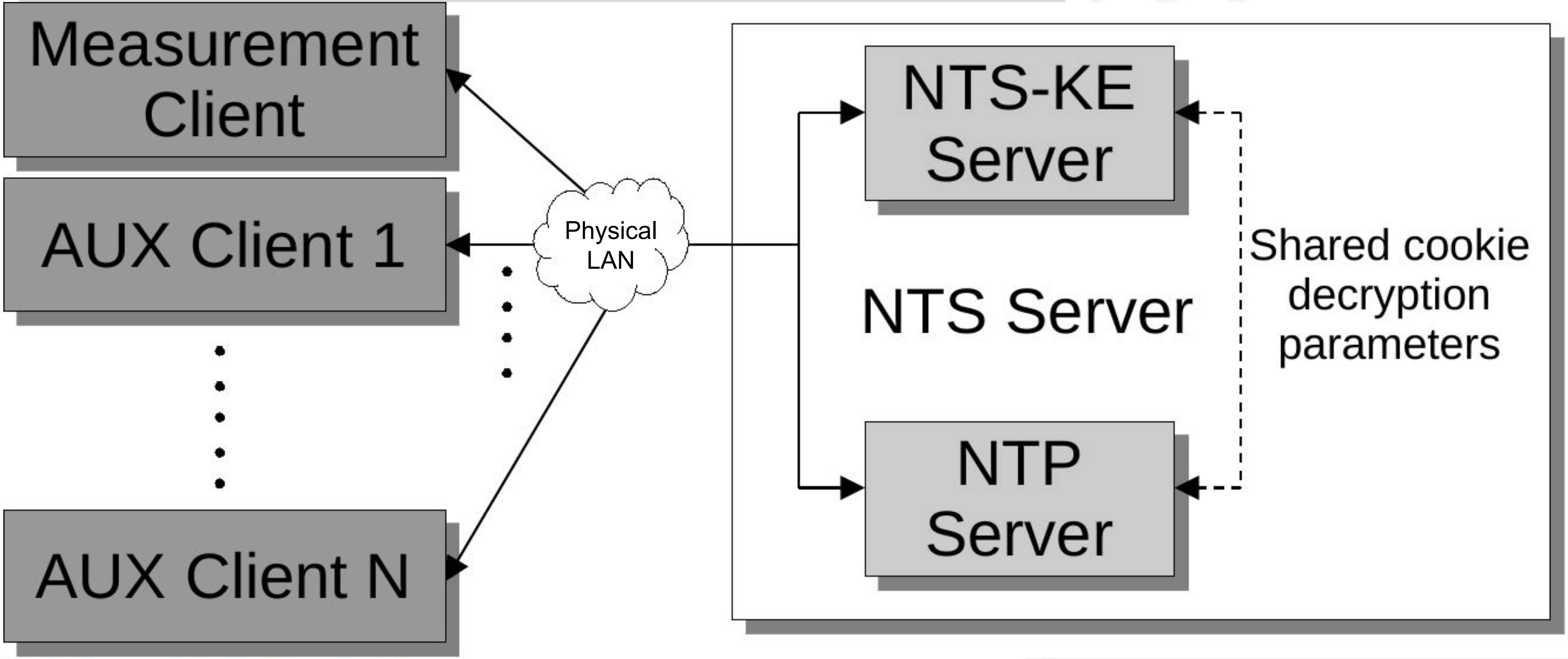
- NTS-KE adds an overhead of 18.13 μs
- Repeated server side operations increased from 2.03 μs to 80.80 μs

Scalability Study

Determines how many requests per second (rps) the NTS-KE and the NTP server could process

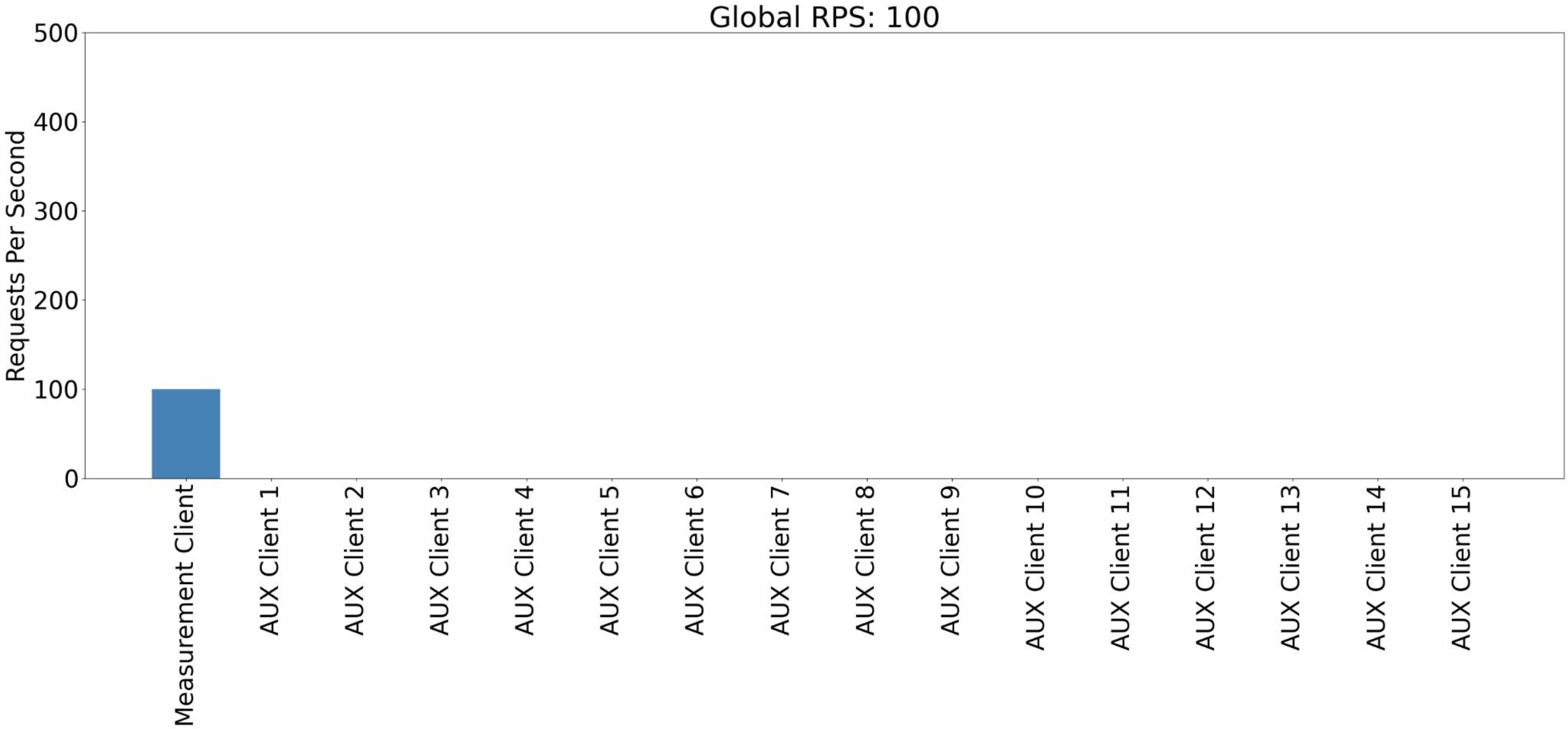
Multiple client machines were used to issue a high number of rps

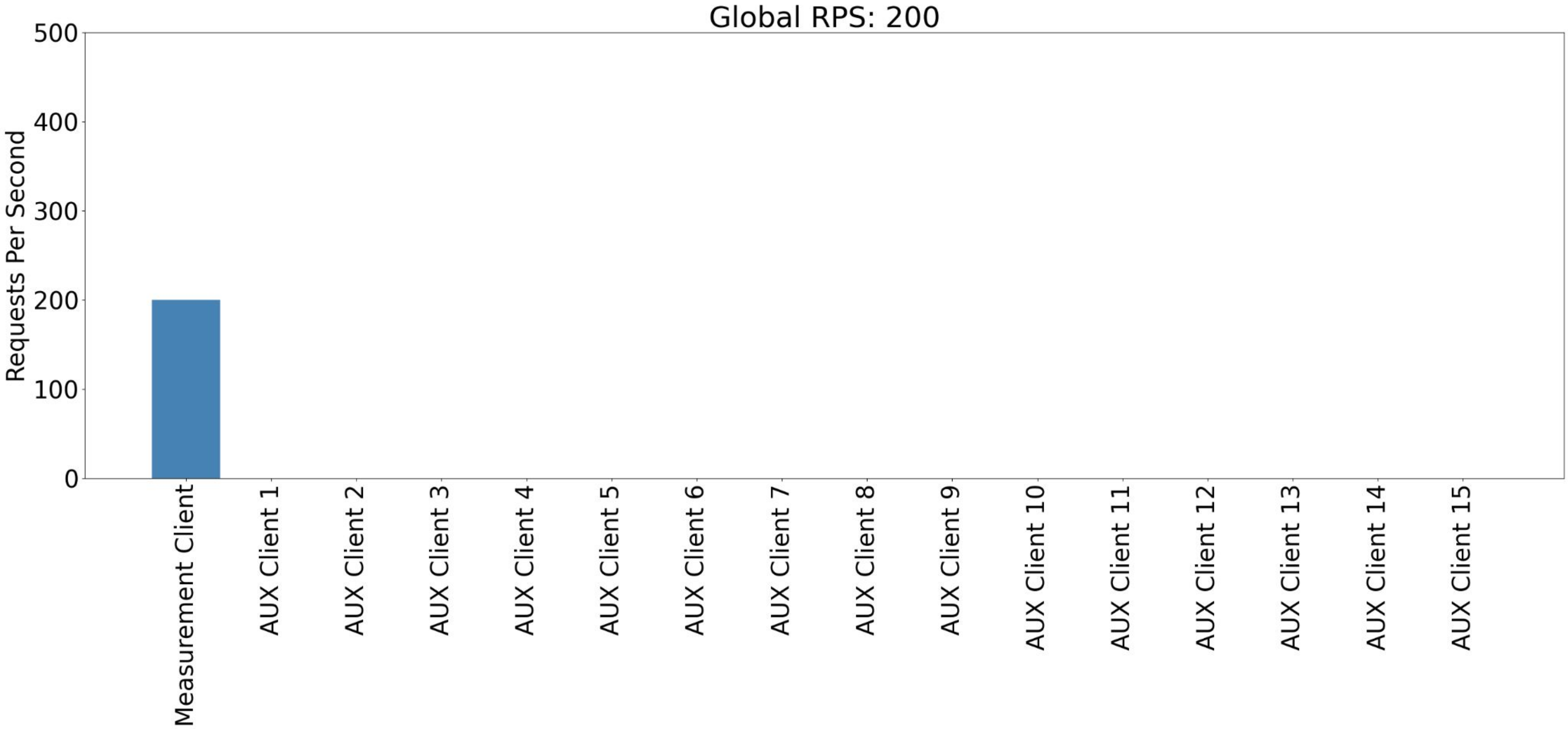
Scalability Experiment Topology

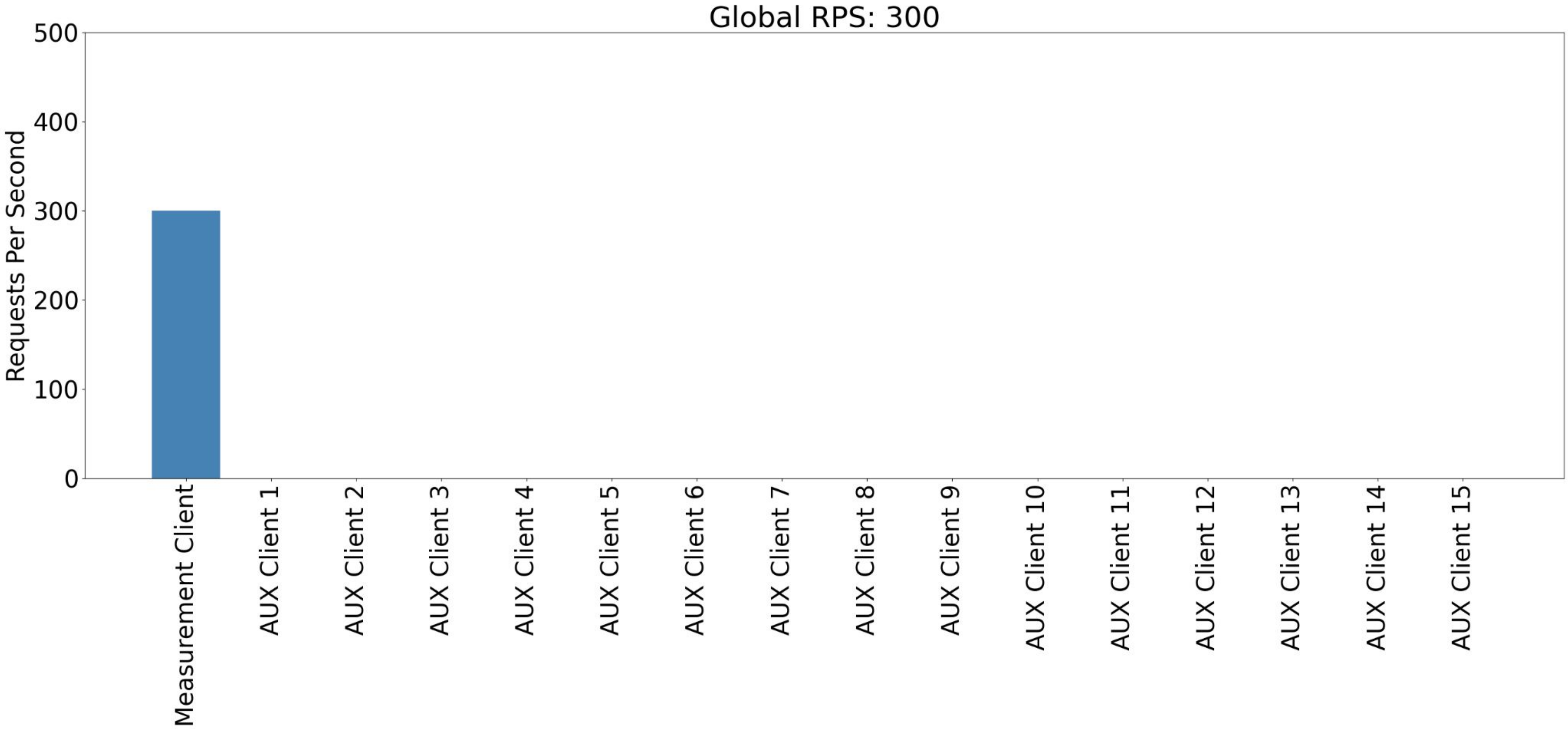


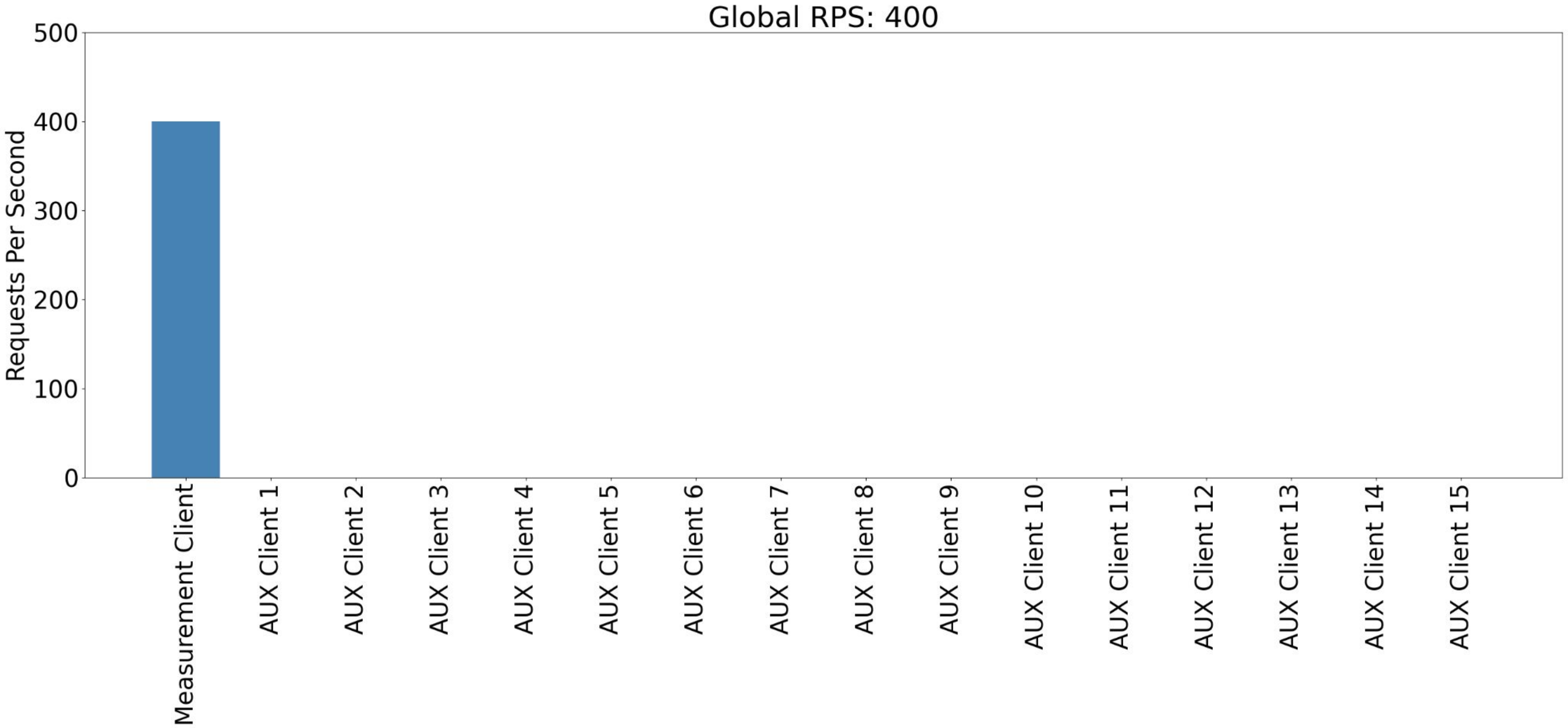
Scalability Description

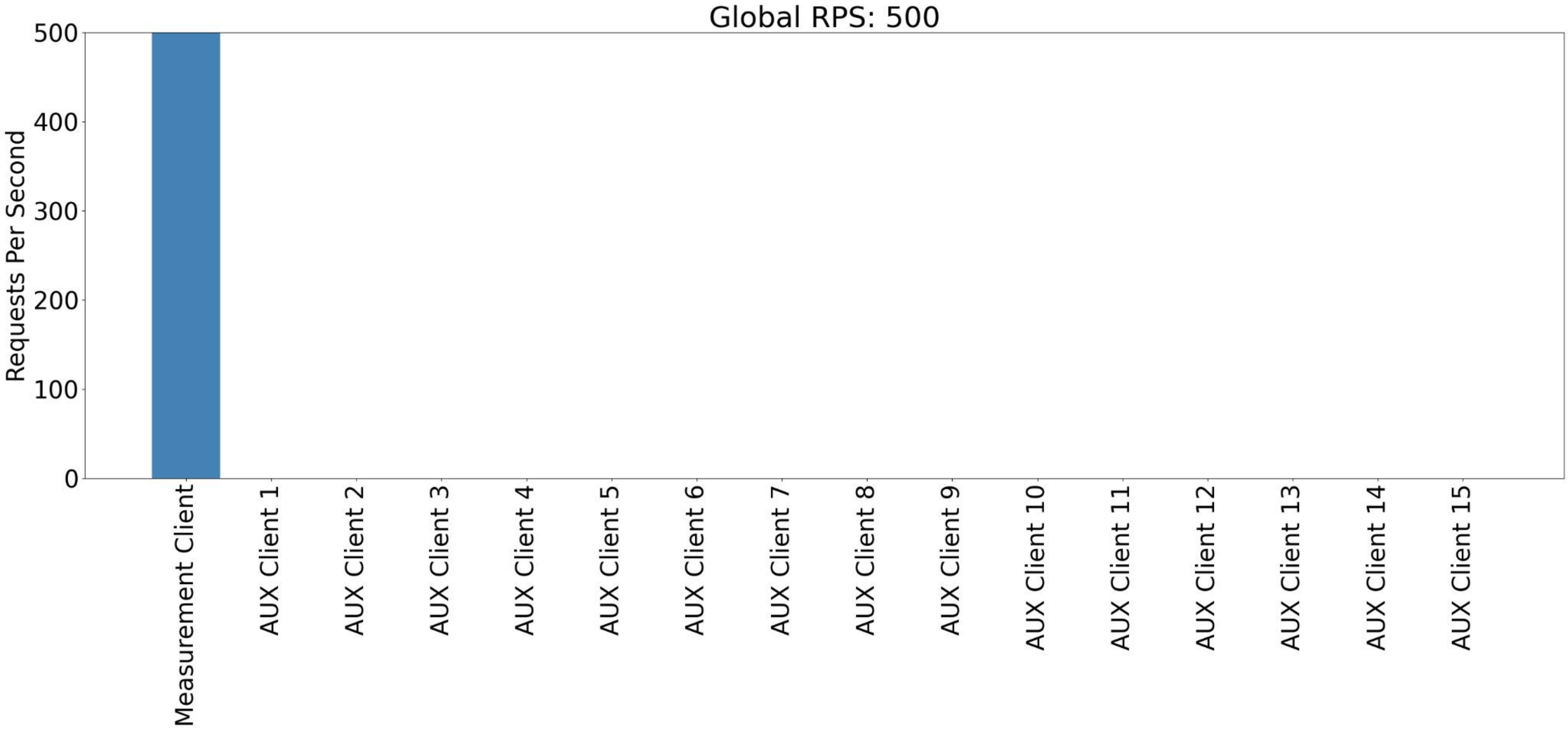
1. Measurement client issues 100 rps for 20 seconds
2. Increases the number of requests by 100 each iteration until 500 rps
3. After gathering measurements at 500 rps an AUX client is enabled and begins issuing 500 rps and the measurement client begins again at 100 rps, resulting in a global load of 600 rps
4. This pattern continues until 8000 rps are issued globally


Global RPS: 100
Requests Per Second
500
400
300
200
100
0
Measurement Client
AUX Client 1
AUX Client 2
AUX Client 3
AUX Client 4
AUX Client 5
AUX Client 6
AUX Client 7
AUX Client 8
AUX Client 9
AUX Client 10
AUX Client 11
AUX Client 12
AUX Client 13
AUX Client 14
AUX Client 15

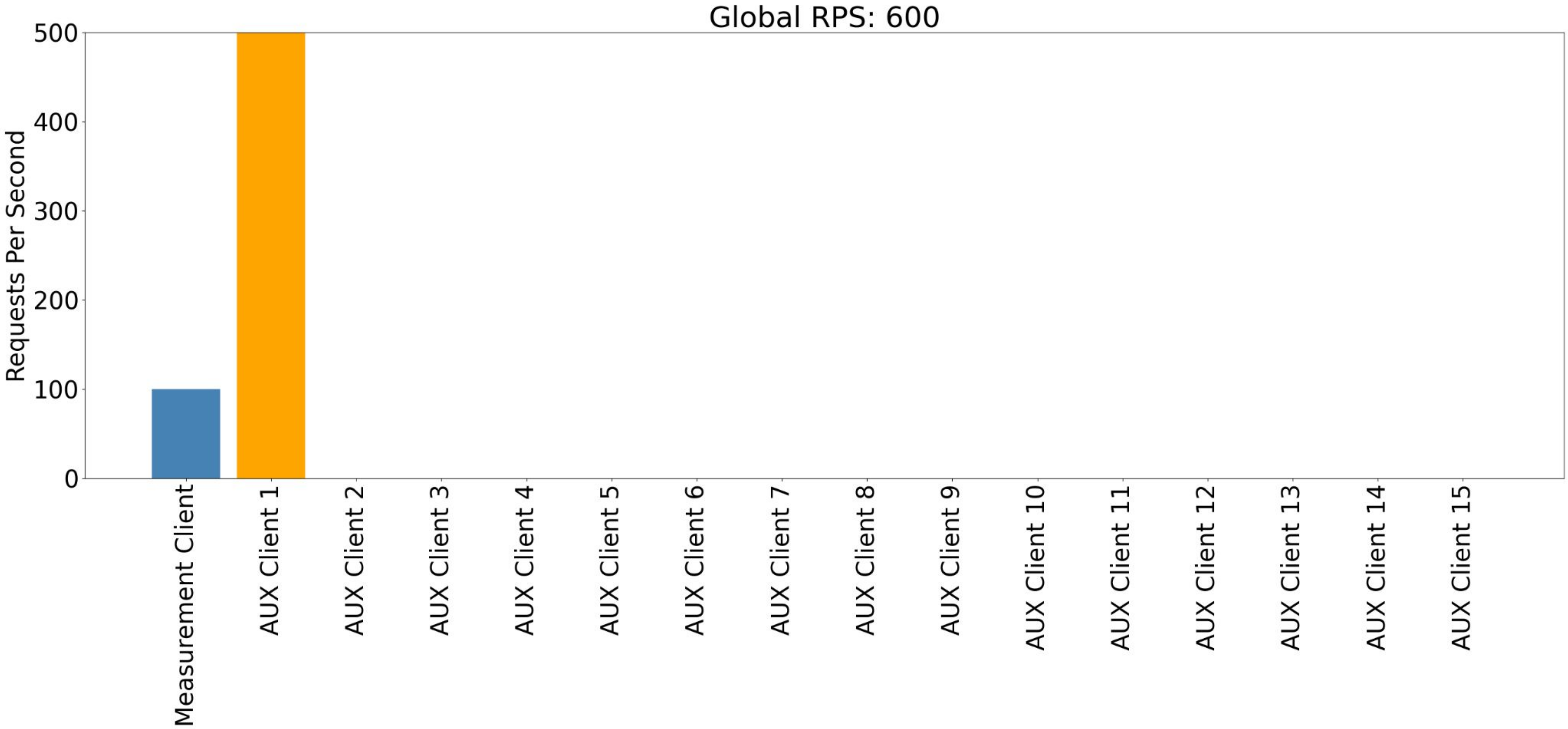

Global RPS: 200
Requests Per Second
500
400
300
200
100
0
Measurement Client
AUX Client 1
AUX Client 2
AUX Client 3
AUX Client 4
AUX Client 5
AUX Client 6
AUX Client 7
AUX Client 8
AUX Client 9
AUX Client 10
AUX Client 11
AUX Client 12
AUX Client 13
AUX Client 14
AUX Client 15


Global RPS: 300
Requests Per Second
500
400
300
200
100
0
Measurement Client
AUX Client 1
AUX Client 2
AUX Client 3
AUX Client 4
AUX Client 5
AUX Client 6
AUX Client 7
AUX Client 8
AUX Client 9
AUX Client 10
AUX Client 11
AUX Client 12
AUX Client 13
AUX Client 14
AUX Client 15

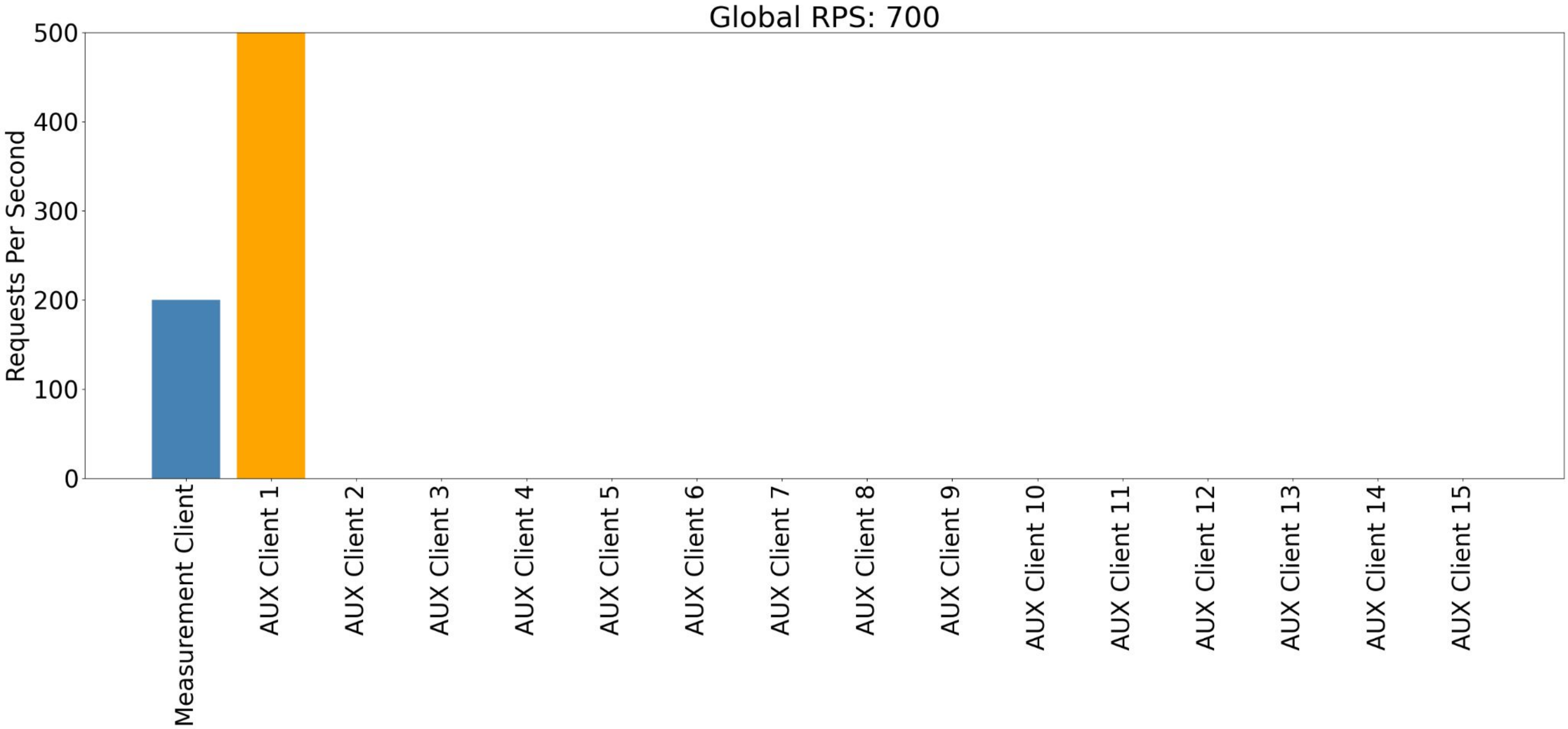

Global RPS: 400
Requests Per Second
500
400
300
200
100
0
Measurement Client
AUX Client 1
AUX Client 2
AUX Client 3
AUX Client 4
AUX Client 5
AUX Client 6
AUX Client 7
AUX Client 8
AUX Client 9
AUX Client 10
AUX Client 11
AUX Client 12
AUX Client 13
AUX Client 14
AUX Client 15

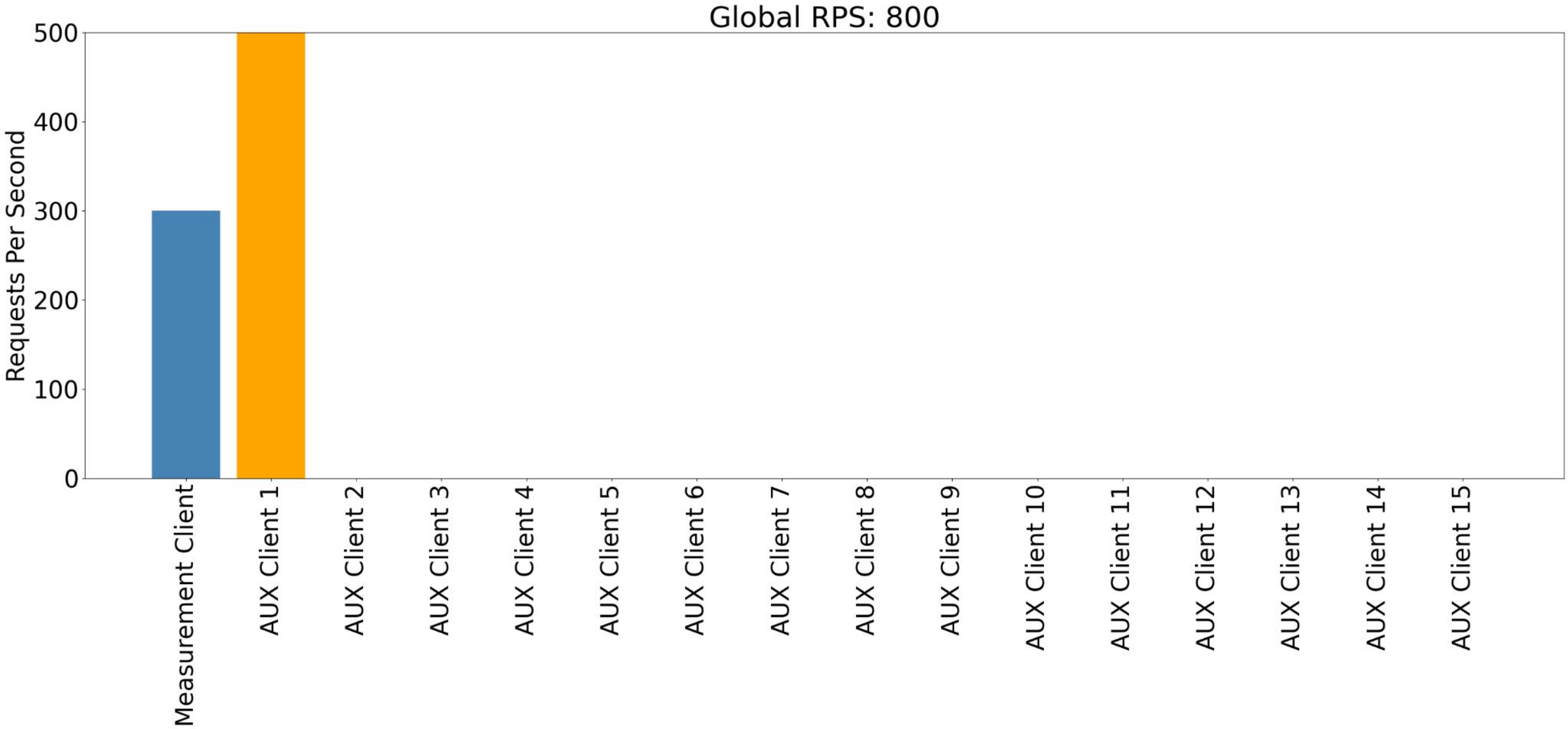


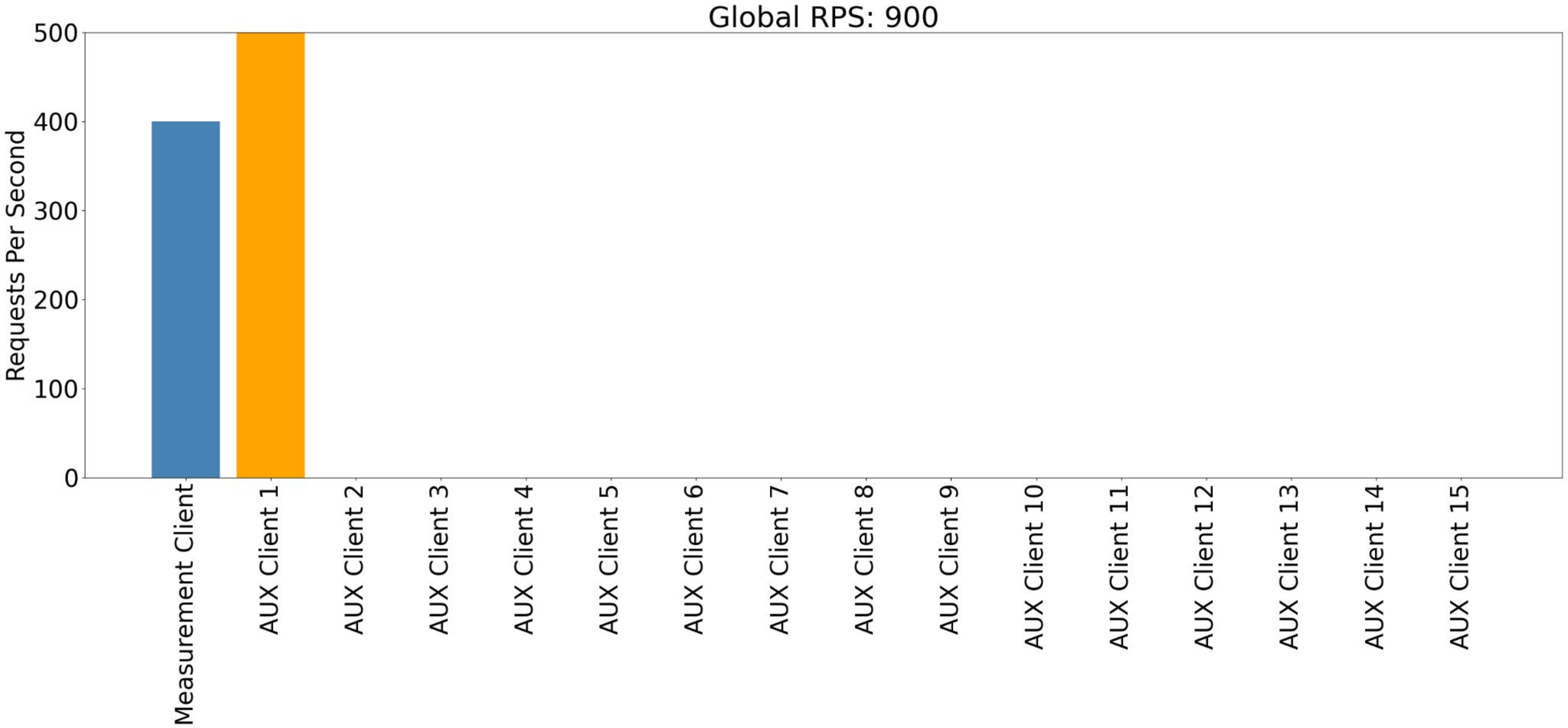
University of New Hampshire InterOperability Laboratory



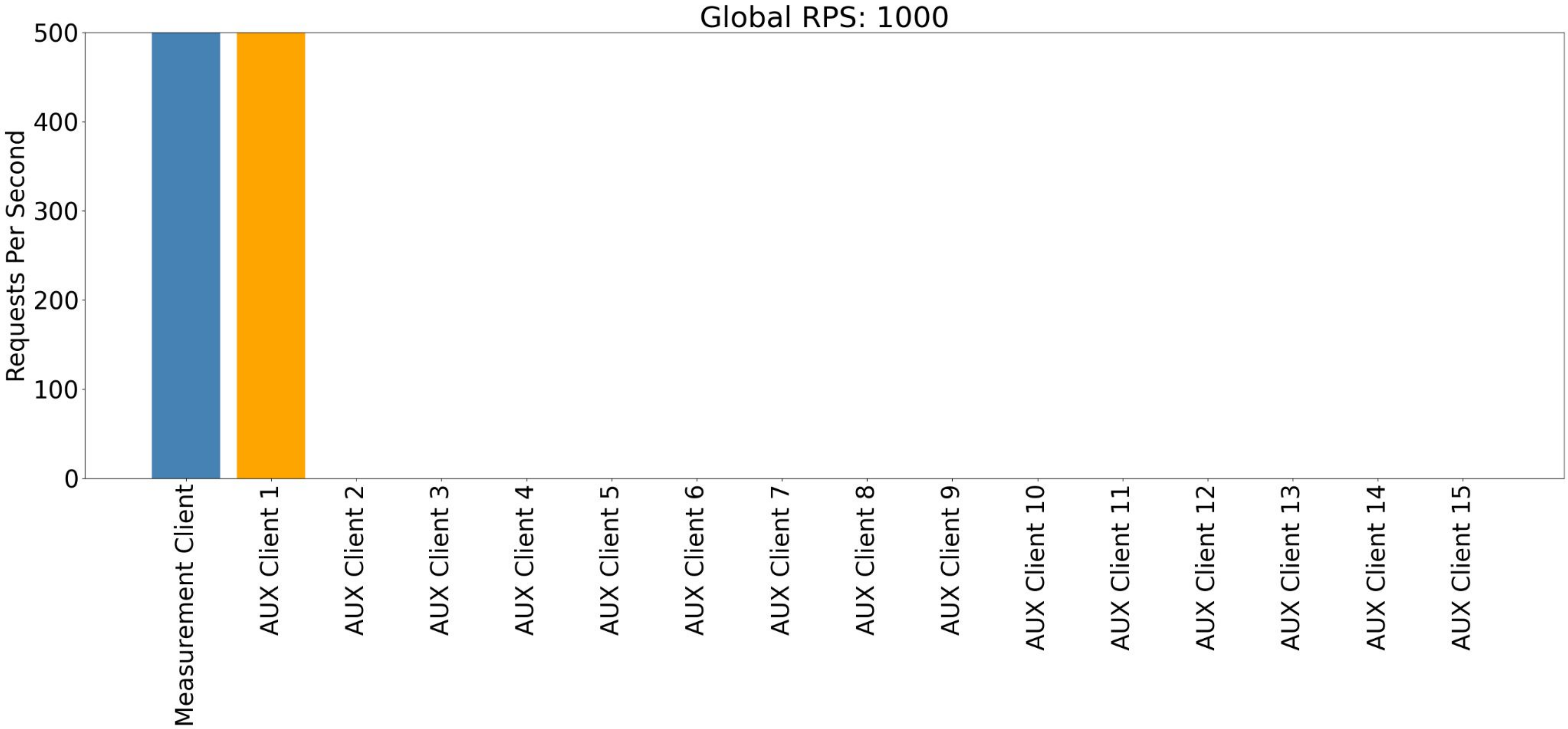
Global RPS: 600
Requests Per Second
500
400
300
200
100
0
Measurement Client
AUX Client 1
AUX Client 2
AUX Client 3
AUX Client 4
AUX Client 5
AUX Client 6
AUX Client 7
AUX Client 8
AUX Client 9
AUX Client 10
AUX Client 11
AUX Client 12
AUX Client 13
AUX Client 14
AUX Client 15


Global RPS: 700
Requests Per Second
500
400
300
200
100
0
Measurement Client
AUX Client 1
AUX Client 2
AUX Client 3
AUX Client 4
AUX Client 5
AUX Client 6
AUX Client 7
AUX Client 8
AUX Client 9
AUX Client 10
AUX Client 11
AUX Client 12
AUX Client 13
AUX Client 14
AUX Client 15

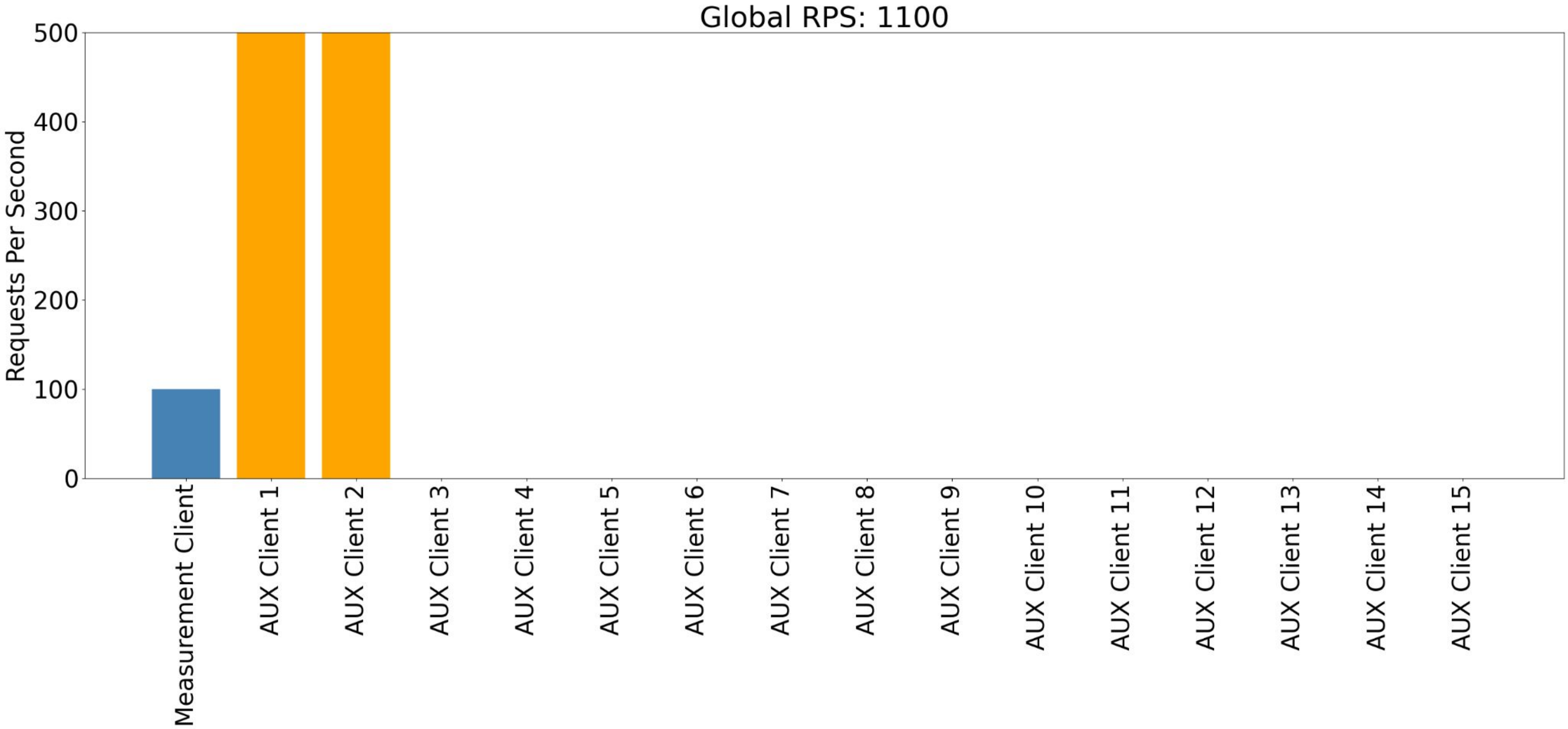

Global RPS: 800
Requests Per Second
500
400
300
200
100
0
Measurement Client
AUX Client 1
AUX Client 2
AUX Client 3
AUX Client 4
AUX Client 5
AUX Client 6
AUX Client 7
AUX Client 8
AUX Client 9
AUX Client 10
AUX Client 11
AUX Client 12
AUX Client 13
AUX Client 14
AUX Client 15



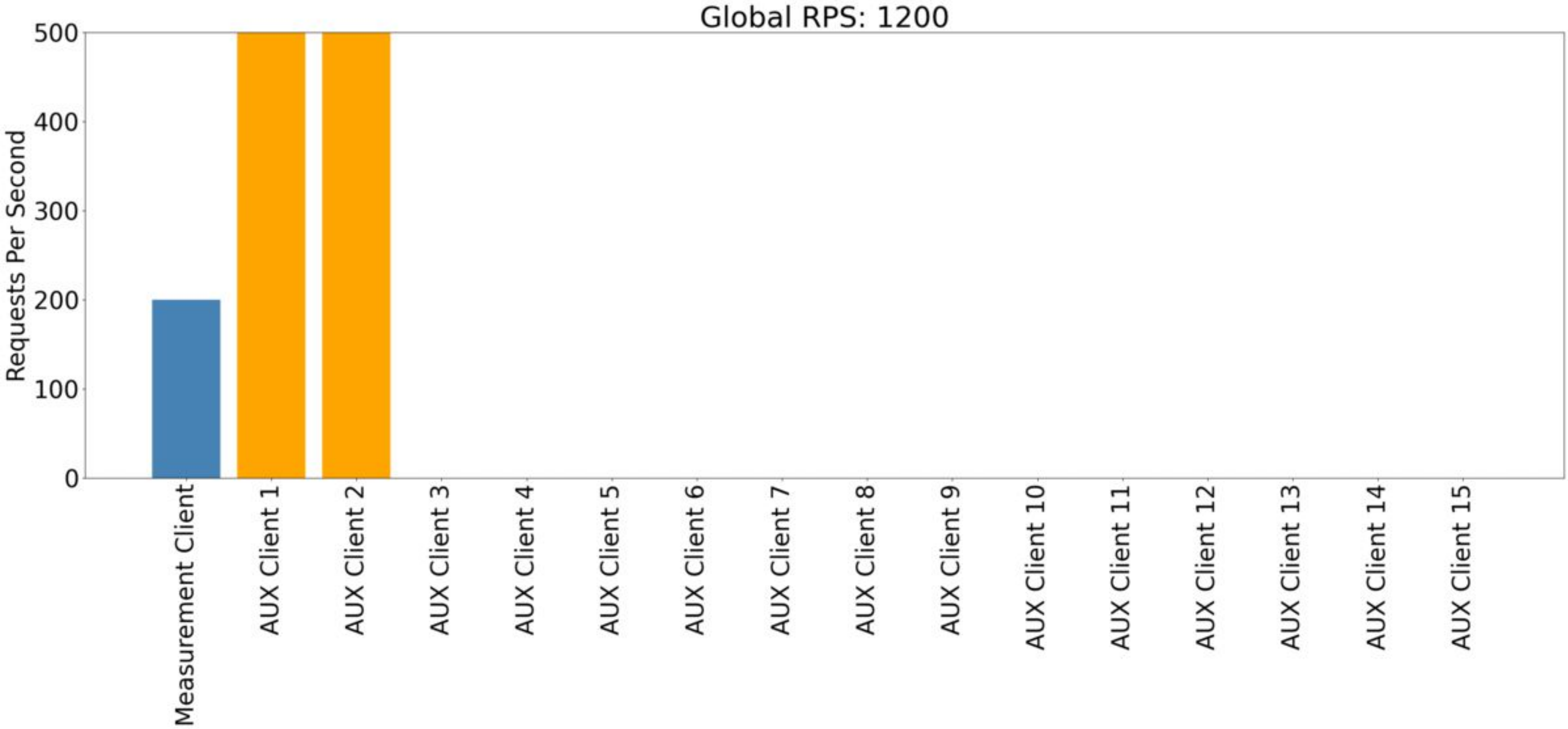
Global RPS: 900
Requests Per Second
500
400
300
200
100
0
Measurement Client
AUX Client 1
AUX Client 2
AUX Client 3
AUX Client 4
AUX Client 5
AUX Client 6
AUX Client 7
AUX Client 8
AUX Client 9
AUX Client 10
AUX Client 11
AUX Client 12
AUX Client 13
AUX Client 14
AUX Client 15



University of New Hampshire
InterOperability
Laboratory



Global RPS: 1100
Requests Per Second
500
400
300
200
100
0
Measurement Client
AUX Client 1
AUX Client 2
AUX Client 3
AUX Client 4
AUX Client 5
AUX Client 6
AUX Client 7
AUX Client 8
AUX Client 9
AUX Client 10
AUX Client 11
AUX Client 12
AUX Client 13
AUX Client 14
AUX Client 15


Global RPS: 1200
Requests Per Second
500
400
300
200
100
0
Measurement Client
AUX Client 1
AUX Client 2
AUX Client 3
AUX Client 4
AUX Client 5
AUX Client 6
AUX Client 7
AUX Client 8
AUX Client 9
AUX Client 10
AUX Client 11
AUX Client 12
AUX Client 13
AUX Client 14
AUX Client 15

Scalability Environment Details

Measurement Client - Virtual machine with eight cores and 16 GB of memory

Auxiliary Client(s) - Virtual machine with four cores and 8 GB of memory

Server - Physical machine with a 4-core 3.3 GHz Intel i5-2500k and 24 GB of memory

- Hosted both the NTS-KE and NTP server

Scalability Experiment Details

The measurement client waits for all active AUX Clients to issue begin issuing requests before measuring response time

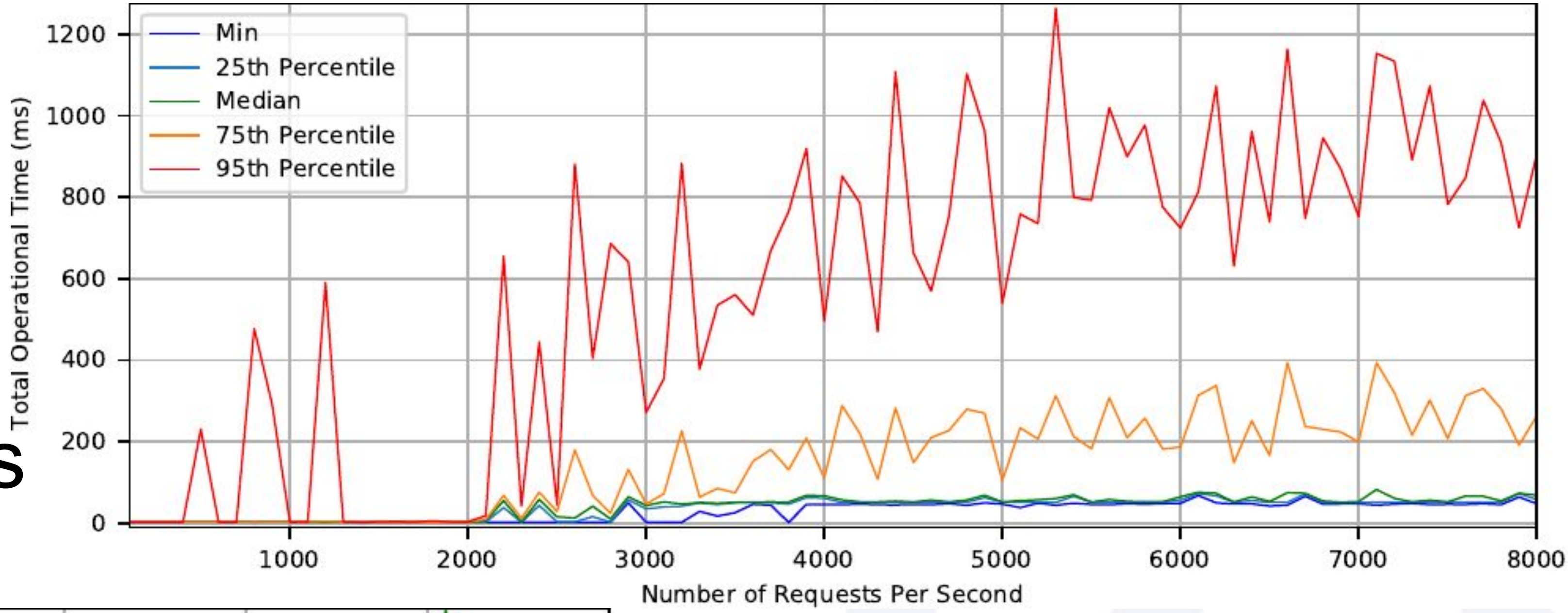
- This emulates a worst-case scenario

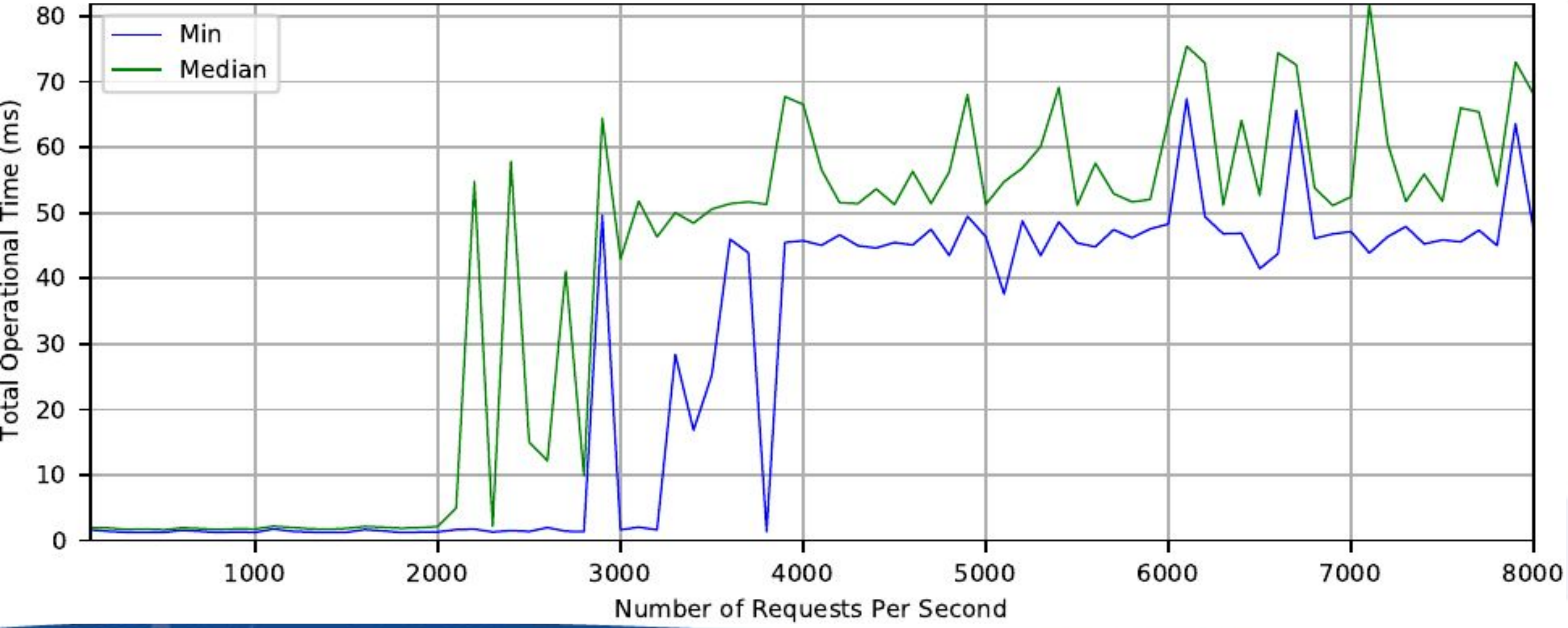
In order to load the NTS-KE and NTP servers, each client was configured to issue three NTPv4 exchanges for each NTS-KE cookie acquired

- This follows the results of the performance study

Beginning at 2000 rps:

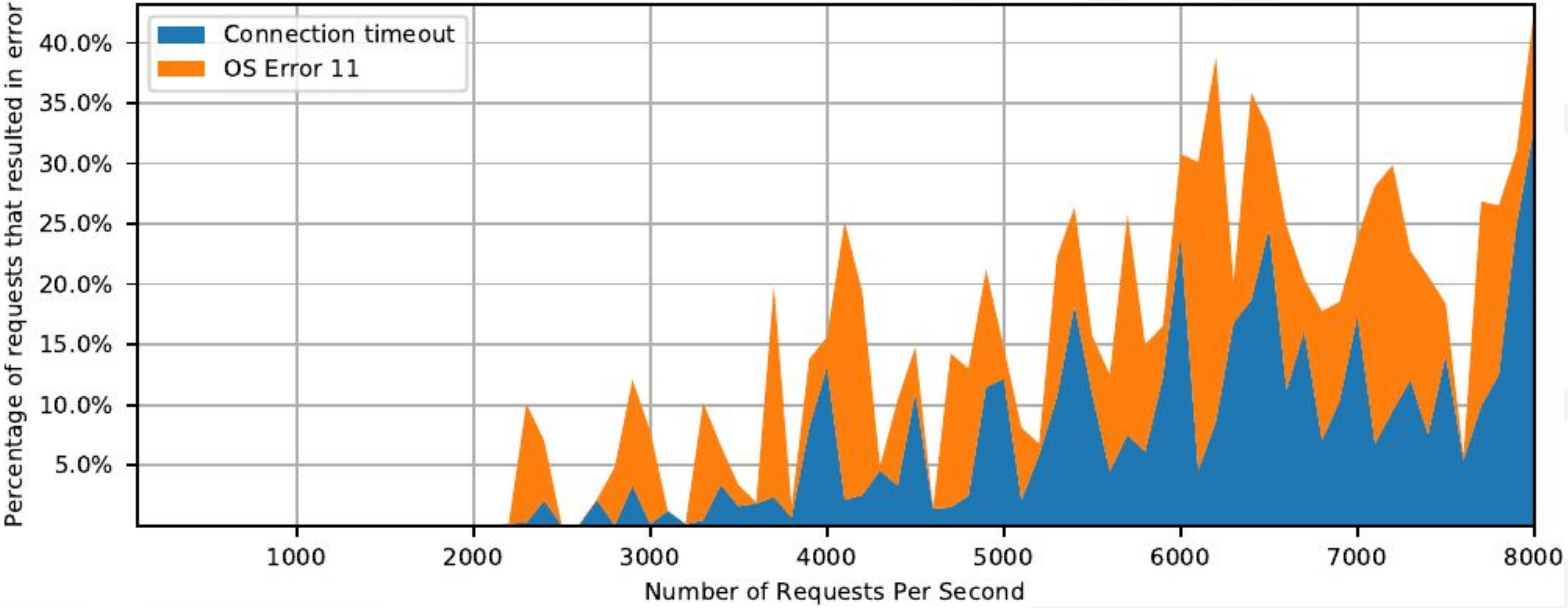
95th percentile d_{CKE} increases by approximately 744.64 ms





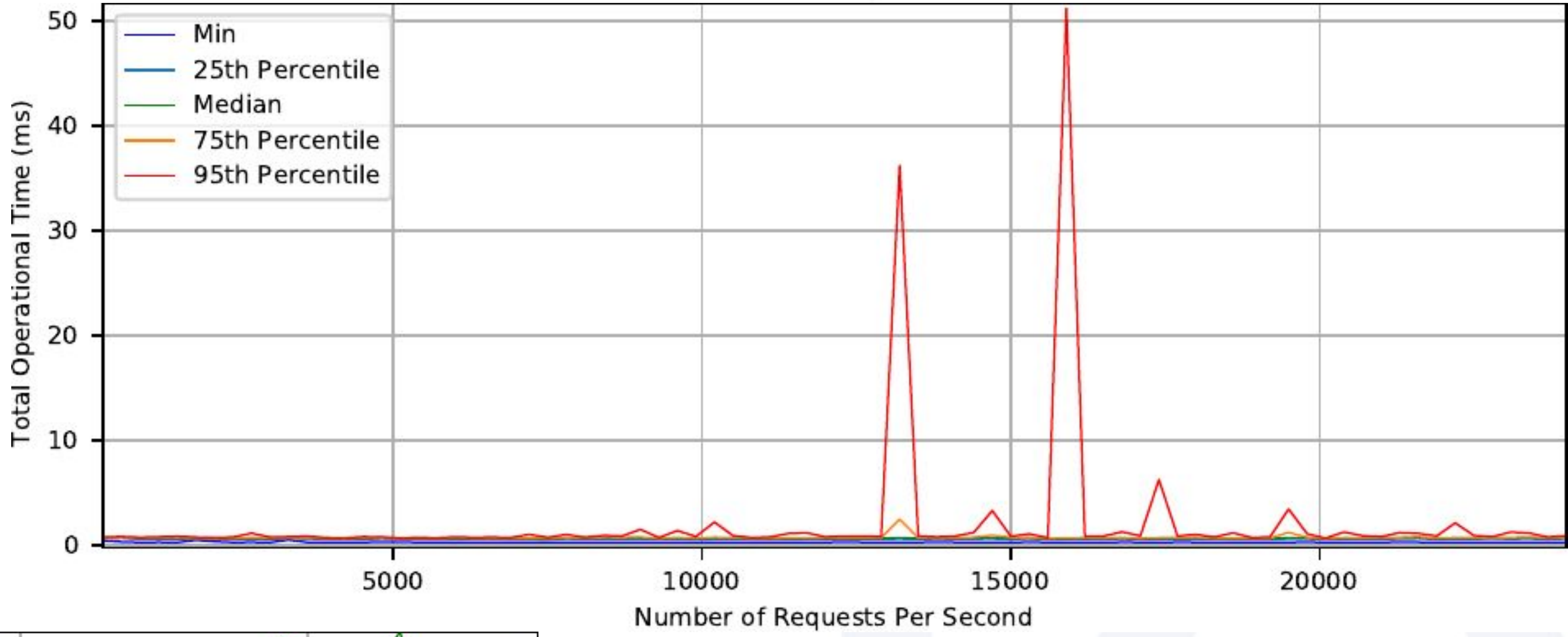
Median d_{CKE} increases by approximately 50.09 ms

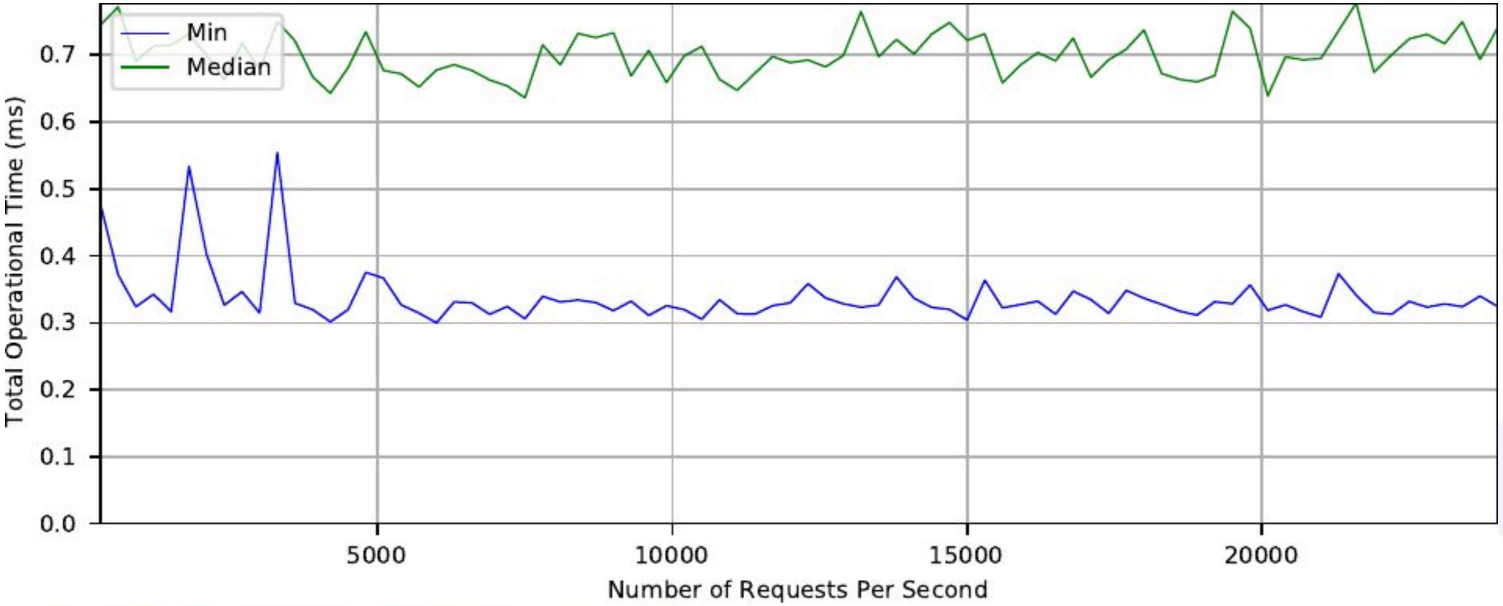
Beginning at 2300 rps:
The client experiences network errors during NTS-KE



University of New Hampshire
InterOperability
Laboratory

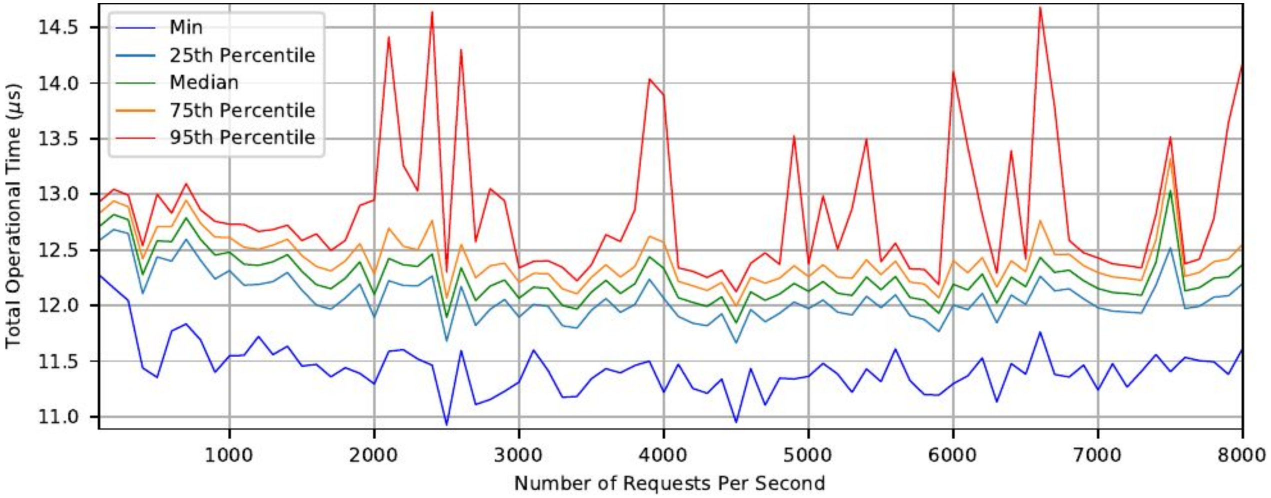
d_{CNTS} remained around the expected value obtained during the performance study of approximately 0.7 ms, with few outliers





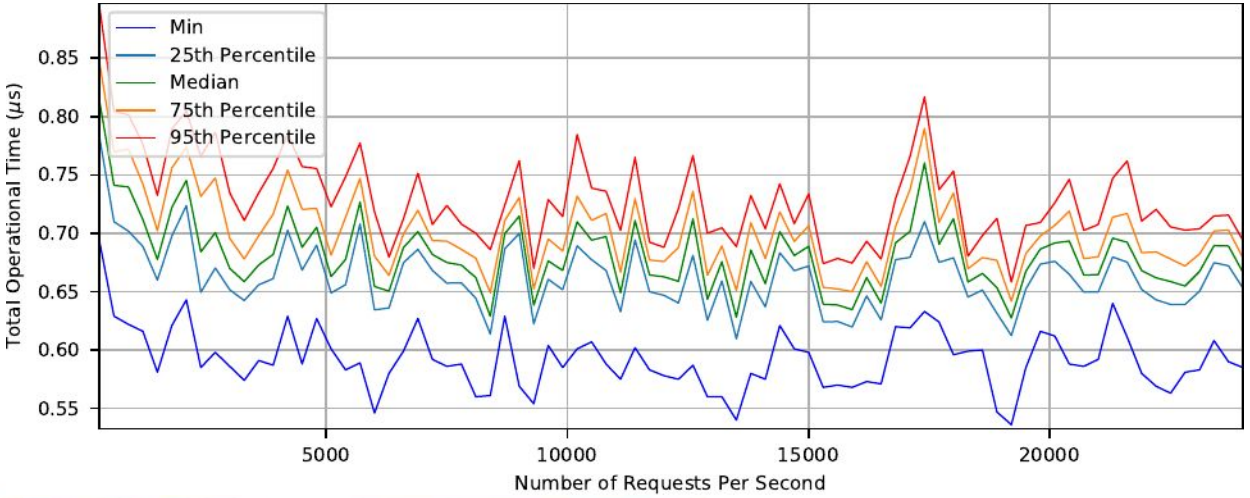
Load was observed to have no effect on authenticated NTPv4 Time transfer

d_{SKE}



All server side operations were unaffected by external network load

d_{SNTP}



d_{SNTS}

Scalability Results

Scalability:

- The NTS-KE server could only process 2000 requests per second before a substantial and consistent increase in response time
- No other measurements were affected by scaling

Conclusions

Performance:

- NTS-KE overhead of 2.26 ms
- Unauthenticated NTPv4 (d_{CNTP}) takes 0.79 ms
- Authenticated NTPv4 (d_{CNTS}) takes 0.87 ms
  - A 9.73% increase
- Repeated server side operations increased from 2.03 μs to 80.80 μs

Scalability:

- The NTS-KE server could only process 2000 requests per second before an increase in response time and error rate
- No other measurements were affected by scaling

University of New Hampshire
InterOperability
Laboratory

Thank You

Griffin Leclerc

University of New Hampshire
InterOperability Laboratory
Department of Computer Science

gleclerc@iol.unh.edu

Radim Bartos

University of New Hampshire
Department of Computer Science

rbartos@unh.edu

www.iol.unh.edu

Reference

RFC 8615

https://datatracker.ietf.org/doc/html/rfc8915

Cloudflare's open source NTS implementation

https://github.com/cloudflare/cfnts

ISPCS 2010

http://archive.ispcs.org/2010/index.html

Implementing Proposed IEEE 1588 Integrated Security Mechanism

https://ieeexplore.ieee.org/document/8543084

Cargo Bench

https://doc.rust-lang.org/unstable-book/library-features/test.html