Just Say NO to Paxos Overhead: Replacing Consensus with Network Ordering

•••

Jialin Li, Ellis Michael, Naveen Kr. Sharma, Adriana Szekeres, and Dan R. K. Ports, University of Washington

Presented by Bartosz Stebel, MIM UW

Outline

1. Basic Paxos refresher
2. Problem statement
3. Ordered Unreliable Multicast
4. NOPaxos
5. Results

Paxos refresher

Paxos is a family of protocols for solving consensus in a unreliable network of unreliable nodes.

- Typically used for state machine replication
- Used in ZooKeeper, Google's Chubby
- Weak network requirements

Asynchronous unreliable networking

- Delivery not guaranteed
- Order not defined
- Latency may be arbitrarily high
- Ubiquitous, cheap, relatively simple to implement
    - Almost every modern network is like this

Paxos refresher continued - basic Paxos

Processor roles:

- **Client** issues requests and waits for responses
- A quorum of **Acceptors** are the fault-tolerant memory
- **Proposers** execute requests on behalf of clients, acting as a coordinator trying to get Acceptors to agree on it
  - One proposer is a **leader**
- **Learners** listen for acceptance events from Acceptors and can take action on them (e.g. send a response to the client)

Paxos refresher continued - basic Paxos

Multi-Paxos with collapsed roles

- If the leader is stable, phase 1 (prepare) can be skipped
  - If the leader fails, the protocol degrades to basic Paxos
- Include an additional round number **I** with each request, incremented by the leader
- The acceptors, proposers and learners can be merged together to form a single role "**Servers**"

Multi-Paxos with collapsed roles - first run

Multi-Paxos with collapsed roles - subsequent runs

Multi-Paxos with collapsed roles

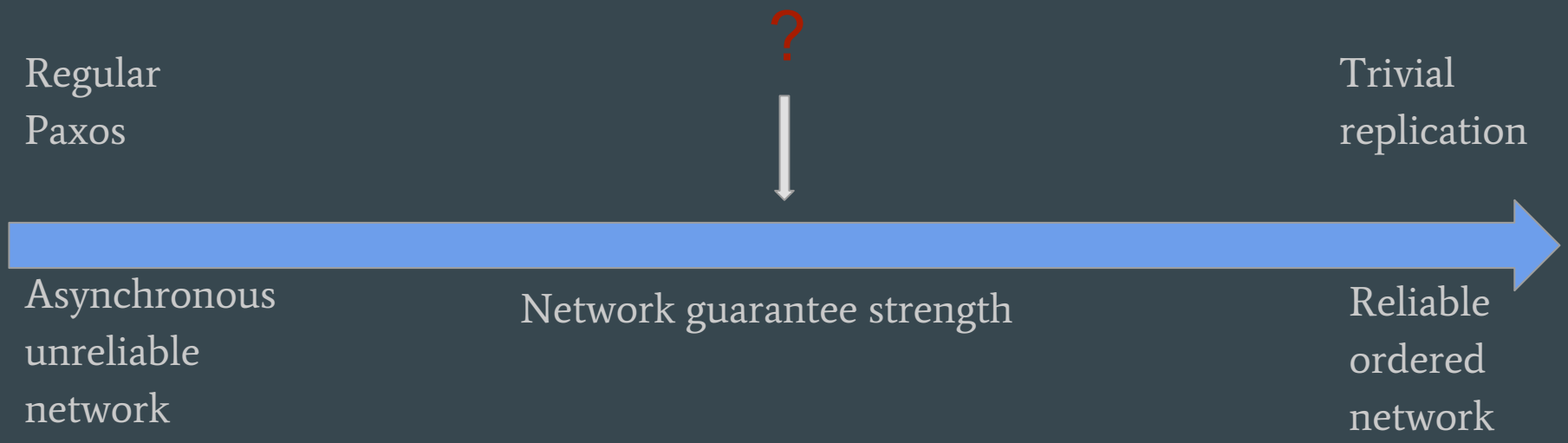
What if we demanded stronger guarantees from the network?

Extreme case: virtually synchronous network

- Reliable delivery
- Consistent ordering
- Replication is trivial in this case

Implementing this model is equivalent to solving the consensus problem - effectively pushes our original problem a layer down the stack

Network-replication complexity tradeoff



A new approach

- New network model: **Ordered Unreliable Multicast**
- New replication protocol: **Network-Ordered Paxos**

Ordered Unreliable Multicast

Let's make the network provide ordering, but leave reliability to the replication protocol

- Route all packets for a certain OUM group via a single **sequencer**
- The sequencer maintains a counter per OUM group and adds a sequence number to each packet
- Guarantees:
  - If two messages, m and m', are multicast to a set of processes, R, then all processes in R that receive m and m' receive them in the same order
  - If some message, m, is multicast to some set of processes, R, then either:
    - Every process in R receives m or a notification that there was a dropped message before receiving the next multicast,
    - No process in R receives m or a dropped message notification for m

Client interface - libOUM library

- Maintains its own counter and emits drop notification events if a skip is detected
- Interface:
  - `send(addr destination, byte[] message)`
  - `getMessage()`
    - Returns the next message, a drop notification or a session terminated error
  - `listen(int sessionNum, int messageNum)`
    - Starts listening for messages in a given session, starting from a given message number

Implementing the sequencer

Several sequencer implementations are possible:

- End-host sequencing
  - Uses conventional servers to run a software implementation
  - Adds significant latency
  - Simple to deploy and implement
- In-switch sequencing
- Hardware middlebox sequencing

In-switch sequencing

Using network switches as sequencers is an ideal solution

- Nearly no latency or bandwidth cost
- Packets can be sequenced while in transit, without a dedicated device
- Requires programmable switch hardware
  - Need to perform flexible per-packet computations
  - Uses the P4 language

In-switch sequencing

A complete implementation in P4 was written, but switches capable of this processing were not available commercially at the time, so performance was not evaluated.

At the time of this presentation, the required hardware is now available, although not very cheap.

e.g. Arista Networks 7170 Series (pictured)

Middlebox prototype

- Since appropriate switches were not available, a prototype was implemented using a Cavium Octeon II CN68XX network processor
  - 32 MIPS64 cores
  - 10 gbit/s Ethernet
- Middle ground between two former options
  - Latency higher than switch-based sequencing, but still considerably lower than end-host
- Processes packets at line rate, so will not be a bandwidth bottleneck

Sequencer fault tolerance

- Routing all packets through the sequencer makes it a single point of failure
- A SDN controller can reroute traffic to a new sequencer if one fails
- Add a *session number* to the total order of messages
  - Incremented each time a sequencer failover occurs
  - Messages from the old and new sequencer are now properly ordered

NOPaxos

Overview

- Utilizes the OUM network primitive
  - Requests are totally ordered, but may be dropped
  - Replicas have to agree which requests to execute and which to ignore, but do not have to agree on the order (which is harder)
- **No coordination** is necessary in the common case
  - Must run consensus if dropped packets are detected

NOPaxos subprotocols

- Normal operation
  - Processing client requests
- Gap agreement
  - Handling dropped packets
- View change
  - Handling leader or network failures
  - Based on the protocol from Viewstamped Replication
- Synchronization
  - Optimization - leader synchronizes the logs of all replicas
  - Allows for faster recovery from leader failure

NOPaxos - normal operation

- No coordination
- 1 round trip time

Where:
N: request-id
V: view-id
S: log-slot-num - index in the log

NOPaxos - Gap Agreement

- If a replica receives a drop notification, it contacts the leader for a copy of the request
- If the leader itself receives a drop notification, it coordinates a commit of a NO-OP operation instead of the original request
  - Inserts a NO-OP into its log at position N
  - Seds a GAP-COMMIT(N) to other replicas
  - Replicas write the NO-OP at N, possibly overwriting and reply to the leader
  - Leader waits for quorum of replies, retrying if necessary
- Clients don't need to be notified - their request will fail as they will not receive a quorum of responses, after which they can retry

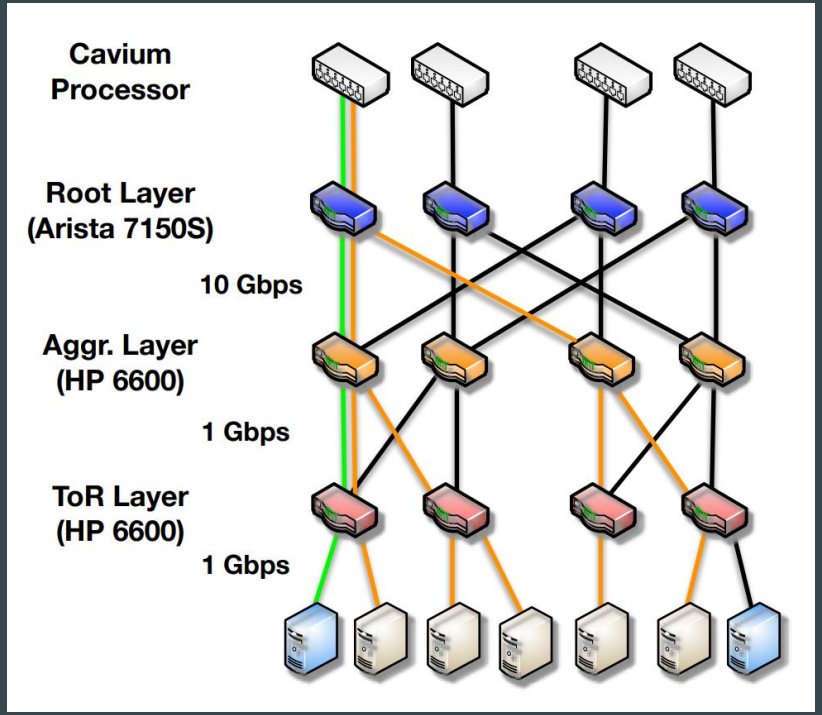
Benefits of NOPaxos

- Achieves the theoretical minimum latency - operations can be executed in one network roundtrip
- Coordination not required in the common case
- Coordination for dropped packets required only if a packet addressed to the leader is dropped
- Each replica (including the leader) sends and receives a **constant** number of messages, irrespective of the total number of replicas

Evaluation results

Evaluation setup

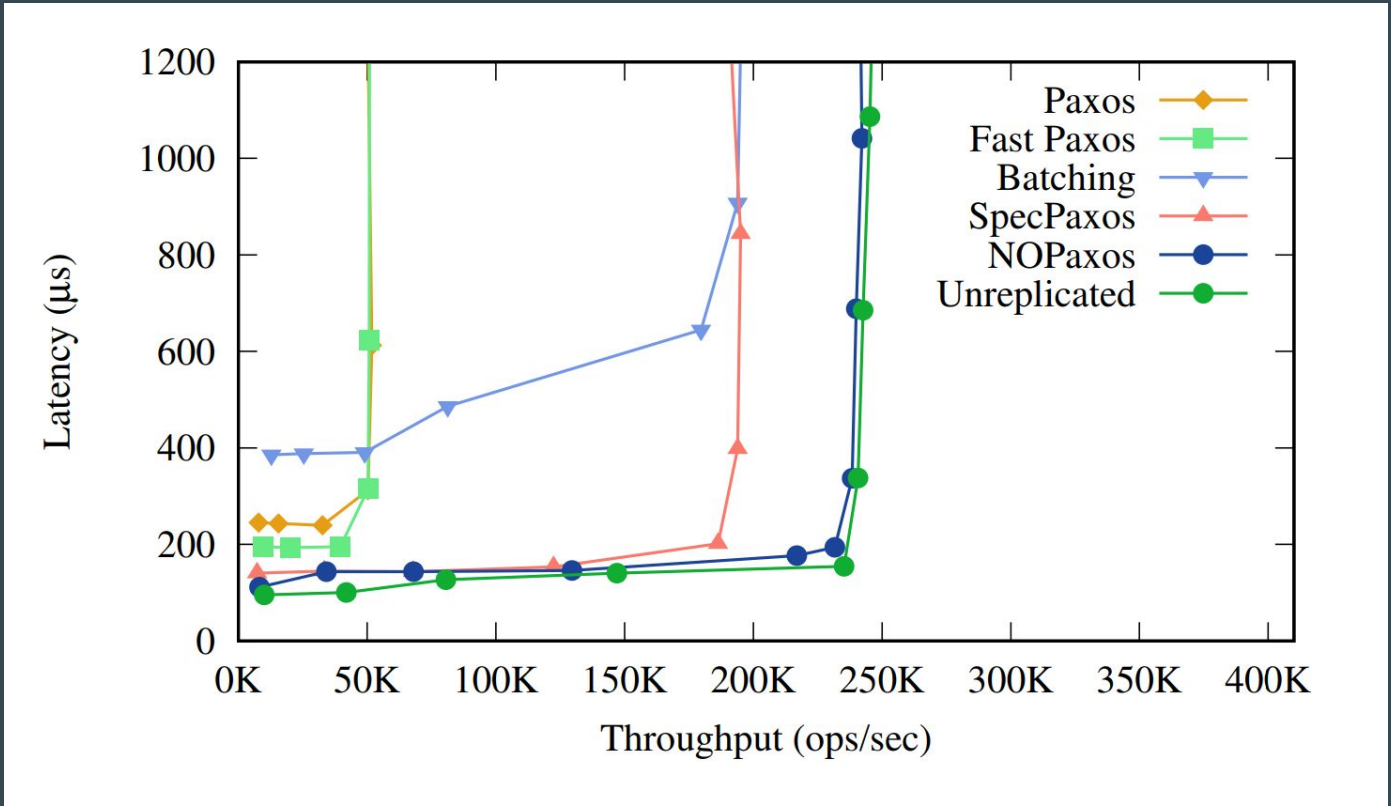
- Protocol implemented in C++
- Network testbed
  - 3 - level fat tree
  - Middlebox sequencer connected to root layer



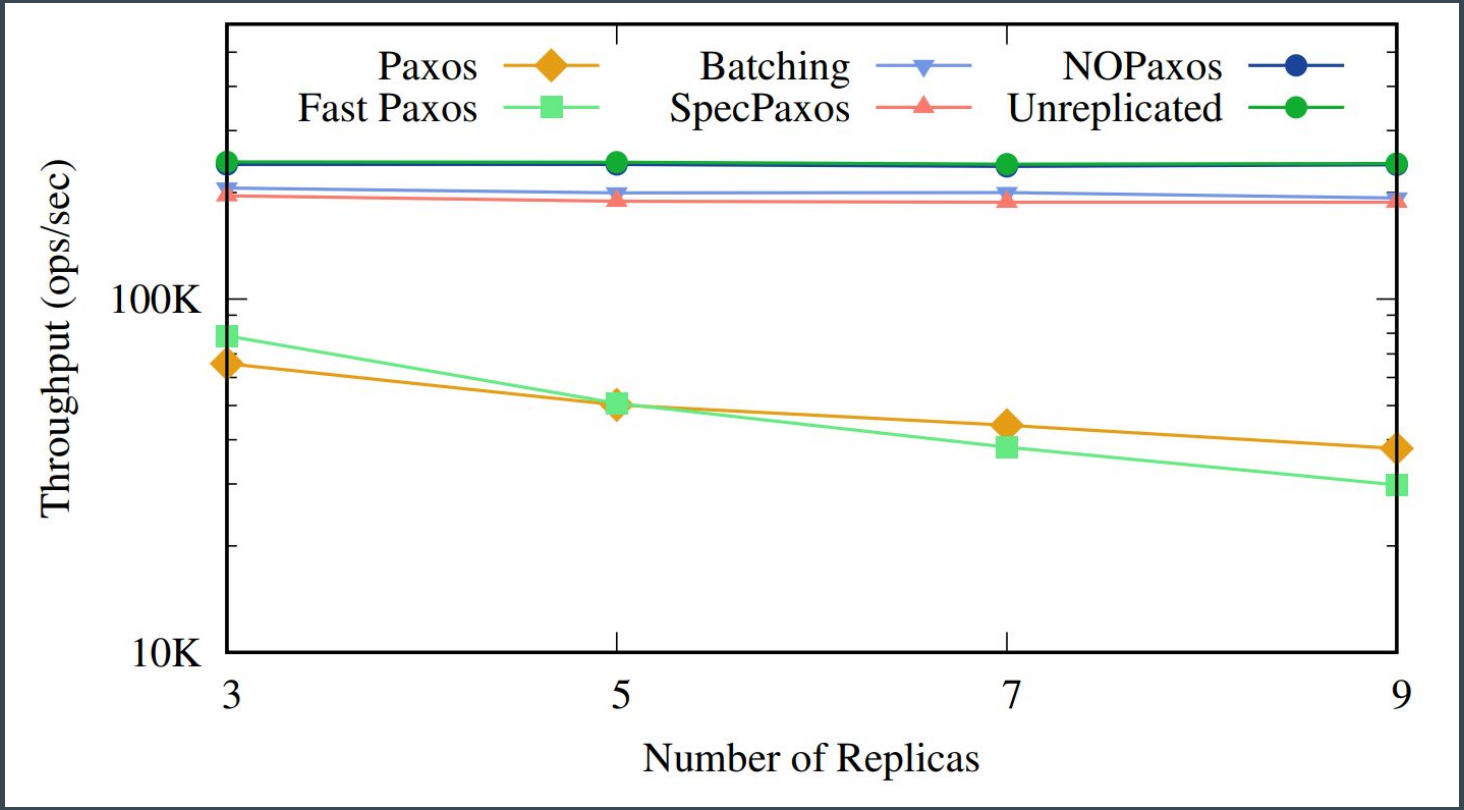
NOPaxos improves both throughput and latency

- 4.7x throughput improvement over Paxos and Fast Paxos
- 40% lower latency than Paxos and Fast Paxos
- 1.25x throughput of Paxos with batching, but with 6x lower latency

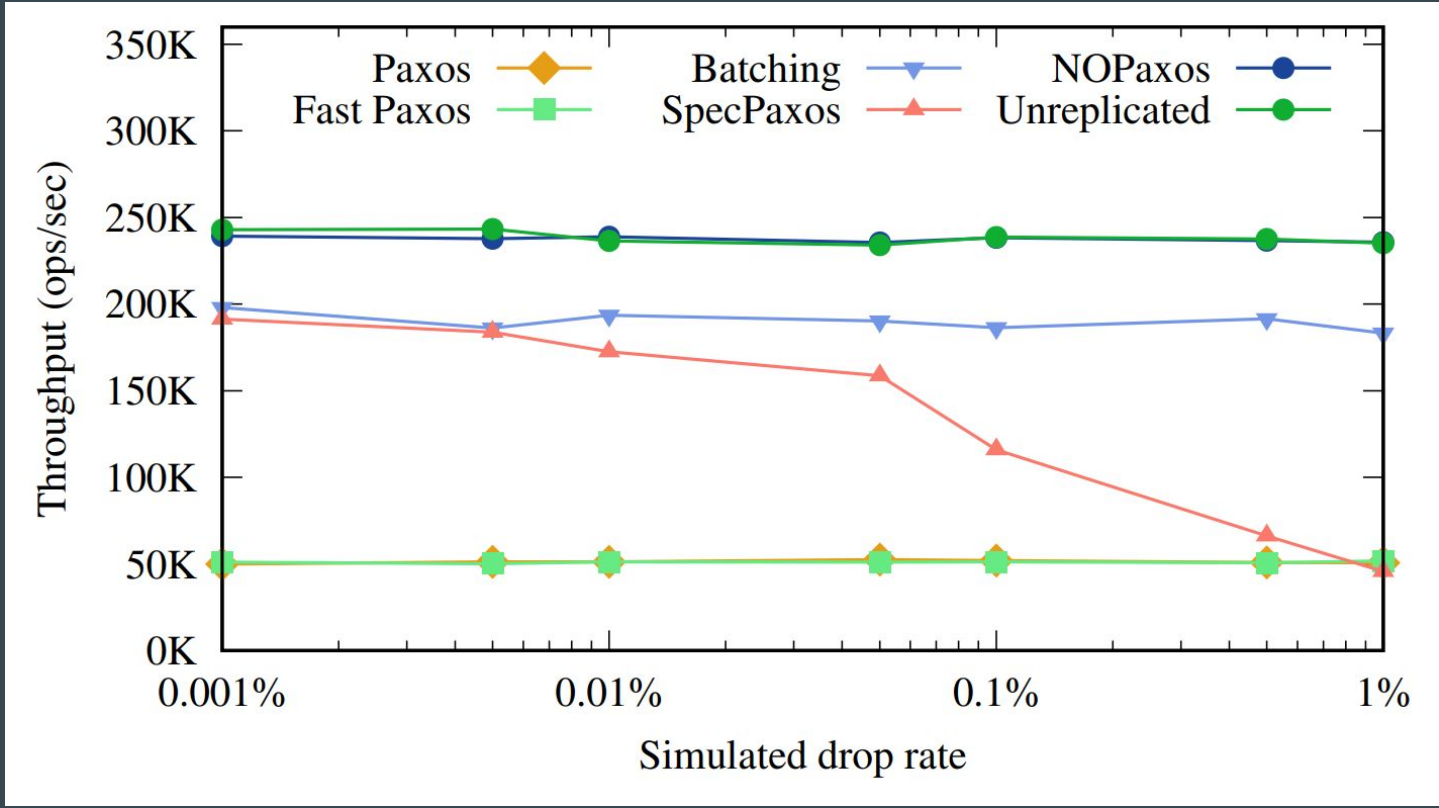
Comparison with other protocols



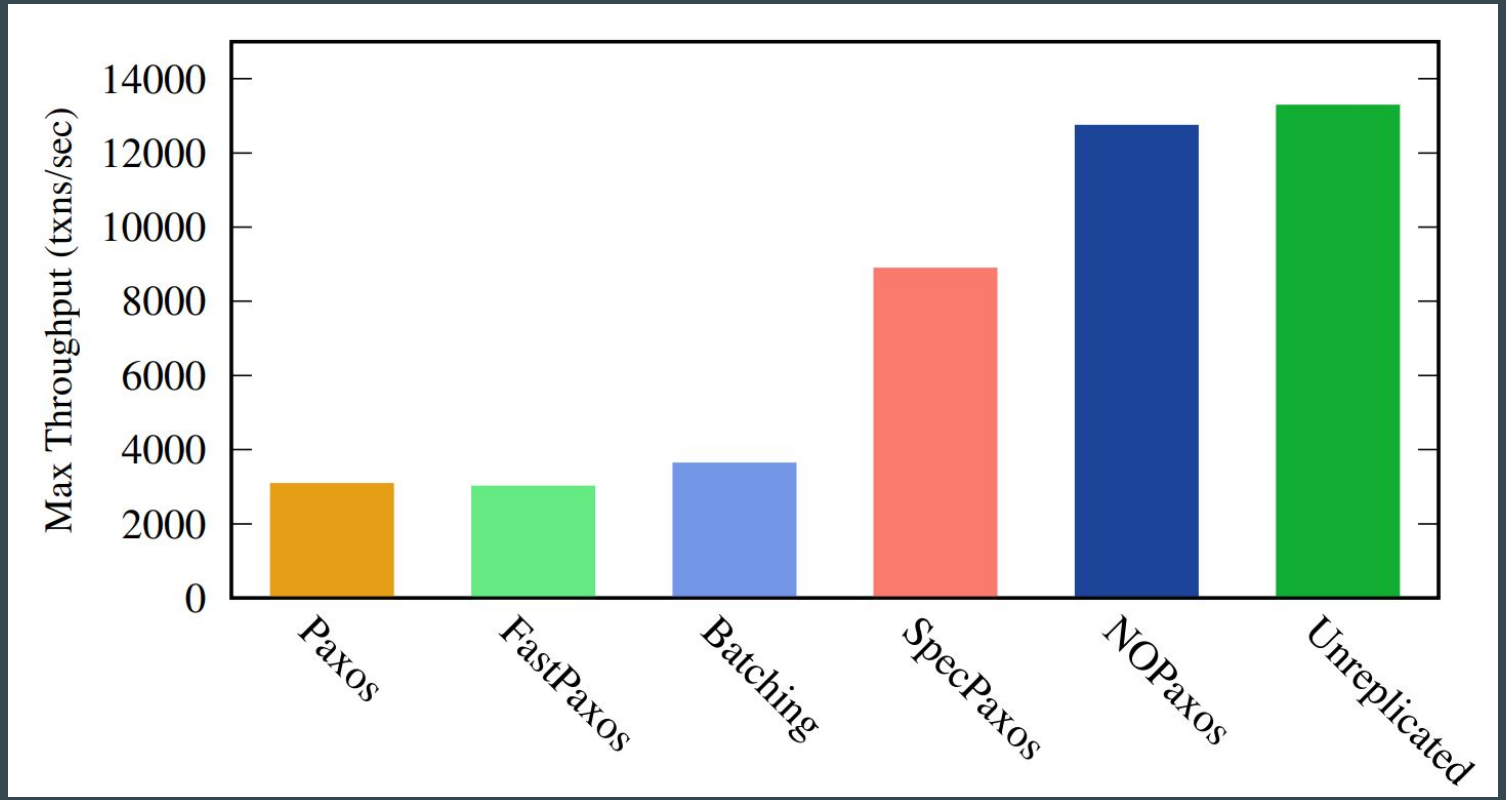
Scaling with number of replicas



Handling of dropped packets



Key-value store benchmark (10ms SLO)



Questions?