Consensus at Network Speed

Huynh Tu Dang, Daniele Sciascia, Pietro Bressana, Han Wang, Ki Suh Lee, Hakim Weatherspoon, Marco Canini, Fernando Pedone, and Robert Soulé

Università della Svizzera italiana (USI),
Cornell University, and Université catholique de Louvain
Consensus is a Fundamental Problem

- Consensus protocols are the foundation for fault-tolerant systems
  - E.g., OpenReplica, Ceph, Chubby
- Many distributed problems can be reduced to consensus
  - E.g., Atomic broadcast, atomic commit
Emerging Technologies

Can we exploit programmable networks for use with consensus?
Outline of This Talk

- Introduction
- **Consensus Background**
- Two Approaches
  - NetPaxos
  - CAANS
- Conclusions
Background
Focus On Paxos

- One of the most widely used consensus protocols
- Lots of prior work on optimization that we can leverage
- “There are two kinds of consensus protocols: those that are Paxos, and those that are incorrect”, attributed to Butler Lampson
Proposers propose a value via the Coordinator (Phase 2).

Acceptors accept value, promise not to accept any more proposals for instance (Phase 2).

Learners require a quorum of messages from Acceptors, “deliver” a value (Phase 2).
Paxos Functionality and Requirements

**Proposer**
- Issue requests. Craft a message with value.

**Coordinator**

**Acceptor**
- Advocate requests. Add a sequence number to the message.
- Choose a value and provide memory. Execute logic, keep persistent state.

**Learner**
- Provide replication. Return chosen value to the application via a callback.
Can We Improve Consensus Performance?

Enforce particular behavior

Consensus Protocols

Programmable Networks

Push logic into network hardware
**Consensus / Network Design Space**

- **Hardware Support**
  - Weak
  - Strong

- **Assumptions**
  - No lost messages
  - FIFO Delivery
  - Best effort
  - Exploit better service guarantees

- **Traditional Paxos**
  - Forward packets
  - Add sequence numbers
  - Persistent storage

- **Implement Paxos in the network**
  - Logic and stateful forwarding
NetPaxos: Weak Hardware Support, Strong Network Assumptions
Exploit Ordering Guarantees: No Sequence Numbers

If messages always arrive in correct order, we don’t need sequence numbers.

Replace coordinators with switch to serialize proposals.
Exploit Ordering Guarantees: No On-Device State

If messages always arrive in correct order, we don’t need on-device state.

Acceptors forward messages to external storage.
Exploit Ordering Guarantees: No On-Device State

If messages always arrive in correct order, we don’t need on-device state.

Acceptors send messages to external storage. Requires additional acceptor to reach consensus.
Exploit Ordering Guarantees: No Acceptor Logic

Acceptors always “accept”. We call them “minions”.

If messages always arrive in correct order we don’t need acceptor logic.
If Assumptions Don’t Hold: Performance Impact

If messages from serializer to minion are not in the same order:

“learner indecision”. Run a round of regular Paxos, consult storage. A-OK!
If Assumptions Don’t Hold: Correctness Impact

If messages from minion to servers are not in the same order:

Experiments

Focus on two questions:

- Do our ordering assumptions hold?
- What is the *potential* benefit of NetPaxos?

Testbed:

- Three Pica8 Pronto 3290 switches, 1Gbps links
- Send messages of 1 MTU size, with sequence numbers
Assumptions Mostly Hold

- **Performance assumption (indecision)** held up to 70% link capacity
- **Correctness assumption (disagreement)** was never violated
- **Traffic should not be bursty**
High Potential for Performance

Disclaimer: Best case scenario

- 9x increase in throughput
- 90% reduction in latency
NetPaxos Summary

- Reduced latency: fewer “true” network hops, including switches
- Increase throughput: avoids potential bottlenecks, reduced logic
- Assumptions may be too strong for practical deployments
- Almost assuming away the need for Paxos
Consensus / Network Design Space

- Assumptions
  - No lost messages
  - FiFO Delivery
  - Best effort

- Exploit
  - Better Service Guarantees

- Traditional Paxos
  - Forward packets
  - Add sequence numbers
  - Persistent storage

- Hardware Support
  - Weak
  - Strong Logic and stateful forwarding

- Implement Paxos in the network
CAANS:
Strong Hardware Support,
Weak Network Assumptions
Design Goals 1: Be a Drop-In Replacement

- Istvav et al. [NSDI ’16] implement ZAB in an FPGA, but require that the application also be implemented in the FPGA.
- High-level languages make hardware development easier.
- Implementing LevelDB in P4 might still be tricky....
Standard Paxos API

```c
void submit(struct paxos_ctx * ctx,
            char * value,
            int size);

void (*deliver)(struct paxos_ctx* ctx,
                int instance,
                char * value,
                int size);

void recover(struct paxos_ctx * ctx,
              int instance,
              char * value,
              int size);
```
Standard Paxos API

```c
void submit(struct paxos_ctx * ctx,
            char * value,
            int size);

void (*deliver)(struct paxos_ctx* ctx,
                int instance,
                char * value,
                int size);

void recover(struct paxos_ctx * ctx,
              int instance,
              char * value,
              int size);
```

- **Send a value**
- **Deliver a value**
- **Discover prior value**
Design Goals 2: Alleviate Bottlenecks

Coordinator and acceptors are to blame!
Hardware/Software

- Proposer
- Coordinator
- Acceptors
- Learner
- Proposer
- Coordinator Backup

Facilitates software API
Hardware/Software

Proposer

Facilitates software API

Coordinator

Alleviates bottlenecks

Coordinator Backup

Acceptors

Learners
Hardware/Software

Challenge: map Paxos logic into stateful forwarding decisions

Facilitates software API

Alleviates bottlenecks
Paxos Header Format

- Network devices don’t create messages, only forward them
- Header is union of all Paxos message fields
- Tradeoff: larger instance number allows for pre-initialization; requires more space

```c
header_type paxos_t {
    fields {
        msgtype : 8;
        inst : INST_SIZE;
        rnd : 8;
        vrnd : 8;
        swid : 64;
        value : VALUE_SIZE;
    }
}
```
Implementation

- Source code
  - Proposer and learner written in C
  - Coordinator and acceptor written in P4

- 3 Compilers
  - P4FPGA
  - Netronome Open-NFP
  - Xilinx SDNet

- 4 Hardware target platforms
  - NetFPGA SUME (4x10G)
  - Netronome Agilio-CX (1x40G)
  - Alpha Data ADM-PCIE-KU3 (2x40G)
  - Xilinx VCU109 (4x100G)
P4 Tools

- **P4FPGA**: *P4 to Bluespec to FPGA*. We implemented some code by hand in Bluespec, and optimized by hand (e.g., naive translation produced 6 tables, only needed 3).

- **Xilinx SDNet**: *P4 to PX to FPGA*. We write a Verilog wrapper around SDNet IP block, matching SDNet interface to SUME interface.

- **Netronome OpenNFP**: *Does not support register operations, uses a custom P4 syntax to call actions written in MicroC.*
P4 FPGA

Platform independent packet processing, control flow, and state management

P4

HLIR

OpenSDN BIR \{ Optimization

P4FPGA Runtime + BlueSpec Templates

Verilog

FPGA Firmware

Xilinx, Altera
Experiments

Focus on three questions:

- What is the absolute performance?
- What is the end-to-end performance?
- What is the performance after failure?

Testbed:

- Four NetFPGA SUME boards in SuperMicro Servers
- One Pica8 Pronto 3922 switch, 10Gbps links
Absolute Performance

- Measured on NetFPGA SUME using P4FPGA
- Throughput is over 9 million consensus messages / second (close to line rate)
- Little overhead latency compared to simply forwarding packets
End-to-End Performance

- Application discards result from “deliver” callback
- 2.24x throughput improvement over software implementation
- 75% reduction in latency
- Similar results when replicating LevelDB as application
Performance After Failure

Coordinator failure with software backup

Acceptor failure
Wrapping Up
What I Thought Would be Next

- Strong Hardware Support
  - NetPaxos
  - New Protocol 1
  - New Protocol 2
  - New Protocol 3
- Weak Hardware Support
  - Traditional Paxos
  - CAANS
What Is Next

- We make consensus great again!
- The ball is now in the application developer’s court
- Suggests direction for future work
Outlook

- The performance of consensus protocols has a dramatic impact on the performance of data center applications.
- Moving consensus logic into network hardware results in significant performance improvements.
- Suggests new line of research: don’t optimize the protocol, investigate how to handle lots of consensus messages.
Acknowledgements

Thank you Google! Faculty award helped support this research.

Thank you to Gordon Brebner and Xilinx for donating two SUME boards, providing access to SDNet, performing measurements.

Thank you to Jici Gao, Mary Pham, Bapi Vinnakotam, and Netronome for providing us with a hardware testbed, and support with using their toolchain.
http://www.inf.usi.ch/faculty/soule/netpaxos.html