Consensus as a Network Service

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

Università della Svizzera italiana (USI), Cornell University, and KAUST
Consensus is a Fundamental Problem

- Many distributed problems can be reduced to consensus
  - E.g., Atomic broadcast, atomic commit

- Consensus protocols are the foundation for fault-tolerant systems
  - E.g., OpenReplica, Ceph, Chubby

- Any improvement in performance would have HUGE impact
Key Idea: Move Consensus Into Network Hardware

- This work focuses on Paxos
  - One of the most widely used consensus protocol
  - Has been proved to be correct

Enabling technology trends:

- Hardware is becoming more *flexible*: e.g. PISA, FlexPipe, NFP-6xxx
- Hardware is becoming more *programmable*: e.g., POF, PX, and P4
Outline of This Talk

- Introduction
- Consensus Background
- Design, Implementation & Evaluation
- Conclusions
Paxos Roles and Communication

- Proposers propose values
- A distinct proposer assumes the role of Coordinator
- Acceptors accept a proposal, promise not to accept any other proposals
- Learners require a quorum of messages from Acceptors, “deliver” a value
Design Goals 1: Be a Drop-In Replacement

- István et al. [NSDI ’16] implement ZAB in FPGAs, but require that the application written in the Hardware Description Language.

- 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);
```

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

Coordinator and acceptors are to blame!
Challenge: map Paxos logic into stateful forwarding decisions
NetPaxos: Header Definition & Parser

```c
header_type paxos_t {
    fields {
        msgtype : 16;
        inst : 32;
        rnd : 16;
        vrnd : 16;
        acptid : 16;
        paxosval : 256;
    }
}
```

```c
parser parse_ethernet {
    extract(ethernet);
    return parse_ipv4;
}
parser parse_ipv4 {
    extract(ipv4);
    return parse_udp;
}
parser parse_udp {
    extract(udp);
    return select(udp.dstPort) {
        PAXOS_PROTOCOL: parse_paxos;
        default: ingress;
    }
}
parser parse_paxos {
    extract(paxos);
    return ingress;
}
```
Acceptor Control Flow

Ingress

Drop ➔ is IPv4?

forward_tbl

is Paxos?

round_tbl

Drop ➔ Packet’s rnd >= acceptor’s rnd?

acceptor_tbl

Egress

load acceptor’s rnd stored in registers

Update:
- registers’ states
- ‘msgtype’
- ‘acptid’
- UDP dst port
```plaintext
control ingress {
    if (valid(ipv4)) {
        apply(forward_tbl);
    }
    if (valid(paxos)) {
        apply(round_tbl);
        if (paxos.rnd >= current.rnd) {
            apply(acceptor_tbl);
        }
    }
}
```
Acceptor Control Flow

Ingress

Drop

forward_tbl

Drop

round_tbl

Drop

acceptor_tbl

Egress
// uint16_t rounds_regs[64000];
register rounds_reg {
  width : 16;
  instance_count : 64000;
}

action read_round() {
  // uint16_t current.round = rounds_reg[paxos.inst]
  register_read(current.round, rounds_reg, paxos.inst);
}

table round_tbl {
  actions { read_round; }
  size : 1;
}
Acceptor Control Flow
action handle_2a(learner_port) {
   // rounds_reg[paxos.inst] = paxos.rnd
   register_write(rounds_reg, paxos.inst, paxos.rnd);

   // vrounds_reg[paxos.inst] = paxos.rnd
   register_write(vrounds_reg, paxos.inst, paxos.rnd);

   // values_reg[paxos.inst] = paxos.rnd
   register_write(values_reg, paxos.inst, paxos.paxosval);

   register_read(paxos.acptid, acceptor_id, 0);
   modify_field(paxos.msgtype, PAXOS_2B);
   modify_field(udp.dstPort, learner_port);
}

table acceptor_tbl {
   reads { paxos.msgtype : exact };
   actions { handle_1a; handle_2a };
}
Implementation

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

- **4 Compilers**
  - P4C
  - P4FPGA
  - Xilinx SDNet
  - Netronome SDK

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

- **2 Software target platforms**
  - Bmv2
  - DPDK (work in progress)
# P4 Compilers

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Target</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4C</td>
<td>Software Switch</td>
<td>Supports most of the P4 constructs</td>
</tr>
<tr>
<td>P4@ELTE</td>
<td>DPDK</td>
<td>Does not support register operations. Limits field length to 32 bits</td>
</tr>
<tr>
<td>P4FPGA</td>
<td>FPGAs</td>
<td>Must write modules for unsupported P4 constructs</td>
</tr>
<tr>
<td>Xilinx SDNet</td>
<td>FPGAs</td>
<td>Does not support register operations. Requires a wrapper for the packet stream</td>
</tr>
<tr>
<td>Netronome SDK</td>
<td>Netronome ISAs</td>
<td>Works only with Netronome devices. Custom actions can be written in Micro-C</td>
</tr>
<tr>
<td>Barefoot Capilano</td>
<td>Barefoot Tofino</td>
<td>Tbps switch</td>
</tr>
</tbody>
</table>
Evaluation
Experiment: What is the Absolute Performance?

- Run Coordinator / Acceptor in isolation

Testbed:

- NetFPGA SUME board in a SuperMicro Server
- A Packet generator for offering load
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
Experiment: What is the End-to-End Performance?

- Comparing NetPaxos to a software-based Paxos (Libpaxos)

Testbed:
- 4 NetFPGA SUME boards in SuperMicro Servers
- An OpenFlow-enable 10 Gbps switch (Pica8 P-3922 switch)
End-to-End Performance

- 2.24x throughput improvement over software implementation
- 75% reduction in latency
- Similar results when replicating LevelDB as application
Next Steps

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

“a HUGE wave of consensus messages is approaching”
Questions & Answers
Performance After Failure

![Graph 1: Coordinator failure with software backup](image1)

![Graph 2: Acceptor failure](image2)
End-to-End Experiment

**NetPaxos** Setup

Programmable device

Application Clients

Run Paxos protocol

Application Servers