The Network is The Computer: Running Distributed Services on Programmable Switches

Robert Soulé
Università della Svizzera italiana
and Barefoot Networks
Conventional Wisdom

- The network is “just plumbing”
- Teach systems grad students the end-to-end principle [Saltzer, Reed, and Clark, 1981]
- Programmable networks are too expensive, too slow, or consume too much power
This Has Changed

A new breed of switch is now available:

- They are programmable
- No power or cost penalties
- They are just as fast as fixed-function devices (6.5 Tbps!)*

* Yes, I work at Barefoot Networks.
If This Trend Continues...

Programmable ASICs will replace fixed-function chips in data centers
What Functionality Belongs in the Network?

- Load Balancing
- Firewall
- Congestion Control
Tremendous Opportunity

Fault-tolerance  Key-Value Store  Stream Processing

Run important, widely used distributed services in the network
Tremendous Opportunity

A 10,000x improvement in throughput

[NetPaxos SOSR ’15, P4xos CCR ’16]
Tremendous Opportunity

2 billion queries / second

with 50% reduction in latency

Key-Value Store

[NetCache, NSDI '17]
Tremendous Opportunity

Process 4 billion events per second.

[Linear Road, SOSR ’18]
Key Questions

This sounds good on paper, but...

- How do we actually program network devices? What are the limitations? What are the abstractions?

- What (parts of) applications could or should be in the network? What is the right architecture?

- Given that we are asking the network to do so much more work, how can we be sure that it is implemented correctly?
Leverage emerging hardware...

... to accelerate distributed services...

... and prove that the implementations are correct.
Outline of This Talk

- Introduction
- Programmable Network Hardware
- Co-designing Networks and Distributed Systems
- Proving Correctness
- Outlook
Programmable Network Hardware
What is A Programmable Network?

"If ip.dst is 10.0.0.1, forward out port 1"
What is A Programmable Network?

Data Plane

Source Language

Compiler

e.g., Merlin
[CoNext ’14]

Rules

Controller

e.g., P4FPGA
[SOSR ’17]

Packets

Control Plane

Source Language

Compiler
**Match Action Table**

Data plane programming specifies:
- fields to read
- possible actions
- size of table

Main abstraction for data plane programming

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Match Action Table

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.1</td>
<td>Drop</td>
</tr>
<tr>
<td>10.0.0.2</td>
<td>Forward out 1</td>
</tr>
<tr>
<td>10.0.0.3</td>
<td>Forward out 2</td>
</tr>
<tr>
<td>10.0.0.4</td>
<td>Modify header</td>
</tr>
</tbody>
</table>

Control plane programming specifies the rules in the table.
Match Action Unit

**Match**
- SRAM for exact match
- TCAM for ternary match

**Action**
- Stateless ALU
  - Limited instruction set
  - Arithmetic operations
  - Bitwise operations
- Stateful ALU
  - Counters
  - Meters

**Massively Parallelized:**
- Data Parallelism for performance
- Pipelined stages for data dependencies
Programmable Data Plane

Programmable ASIC Architecture
P4 Language Concepts
Specify header format and how to parse
Specify header format and how to parse

Define tables that match on header fields and perform actions (e.g., modify or drop)
Specify header format and how to parse

Define tables that match on header fields and perform actions (e.g., modify or drop)

Compose lookup tables
Target Constraints

Parser

Match Action

... Match Action

Match Action

Queues and Crossbar

Match Action

... Match Action

Match Action

De-Parse
Target Constraints

Fixed-length pipeline

Parser → Match Action → Match Action → Queues and Crossbar → Match Action → Match Action

... → ... → De-Parser

Match Action → Match Action → Match Action → Match Action → Match Action

... → ... →...
Target Constraints

- Fixed-length pipeline
- Queues and Crossbar
- Limited Memory
Target Constraints

Fixed-length pipeline

Parser

Match Action

Match Action

Match Action

Match Action

Queues and Crossbar

Match Action

Match Action

Match Action

Match Action

De-Parser

... ...

Match Action

Match Action

Match Action

Match Action

Limited Memory

Data and control dependencies

Limited Memory
Observations

- Architecture is designed for speed and efficiency
- Performance doesn’t come for free
  - Limited degree of programmability
  - Not Turing complete by design
- Language syntax and hardware generations may change, but the basic design is fundamental
Co-Designing Networks and Distributed Systems
What Applications Should We Put in the Network?

Monte Carlo Simulation

Fundamental Building Blocks
## Building Blocks For Distributed Systems

<table>
<thead>
<tr>
<th>Building Block</th>
<th>Description</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consensus</td>
<td>Essential for building fault-tolerant, replicated systems</td>
<td>NetPaxos SOSR ’15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P4xos, CCR ’16</td>
</tr>
<tr>
<td>Caching</td>
<td>Maximize utilization of available resources</td>
<td>NetCache, SOSP ’17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NetChain, NSDI ’18</td>
</tr>
<tr>
<td>Data Processing</td>
<td>In-network computation and analytics</td>
<td>Linear Road, SOSR ’18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In submission</td>
</tr>
<tr>
<td>Publish/Subscribe</td>
<td>Semantically meaningful communication</td>
<td></td>
</tr>
</tbody>
</table>
Consensus Protocols

- Get a group of replicas to agree on next application state
- Consensus protocols are the foundation for fault-tolerant systems
  - E.g., OpenReplica, Ceph, Chubby
- Many distributed systems problems can be reduced to consensus
  - E.g., Atomic broadcast, atomic commit
Ways to Improve Consensus Performance

- Enforce particular network behavior
- Push logic into network hardware
- Programmable Networks
- Consensus Protocols
Consensus / Network Design Space

Assumptions

No message loss, FIFO delivery

Best effort

Programmability

Weak

Strong

Forward packets

Storage and logic

Traditional Paxos
Consensus / Network Design Space

No message loss, FIFO delivery

Assumptions

Fast Paxos

Traditional Paxos

Best effort

Weak

Programmability

Strong

Forward packets

Storage and logic
Consensus / Network Design Space

- **Assumptions**
  - No message loss, FIFO delivery
  - Fast Paxos
  - Traditional Paxos

- **Programmability**
  - Weak
  - Strong

- **Forward packets**
  - Storage and logic

- Protocols:
  - Protocol 1
  - Protocol 2
  - Protocol 3
  - Protocol 4
Consensus / Network Design Space

Programmability

Weak

Strong

Assumptions

Best effort

Forward packets

No message loss, FIFO delivery

NetPaxos

Fast Paxos

Traditional Paxos

Protocol 2

Protocol 3

Protocol 4

Storage and logic
Consensus / Network Design Space

No message loss, FIFO delivery

Best effort

Forward packets

Storage and logic

Assumptions

Weak

Strong

Programmability

NetPaxos

99.9% of the time, assumptions held

Fast Paxos

Traditional Paxos

Protocol 2

Protocol 3

Protocol 4

Protocol 1
Consensus / Network Design Space

- **Programmability**
  - Weak
  - Strong

- **Assumptions**
  - No message loss, FIFO delivery
  - Fast Paxos
  - Traditional Paxos

- **Problems**
  - Promising, but 99.9% correct consensus isn’t practical

- **Protocols**
  - Protocol 2
  - Protocol 3
  - Protocol 4

- **Forward packets**
- **Storage and logic**
Consensus / Network Design Space

Programmability

- Weak
- Strong

Assumptions

- No message loss, FIFO delivery
- Fast Paxos
- Speculative Paxos / No Paxos
- Traditional Paxos

Forward packets

Storage and logic

Best effort

NetPaxos

Protocol 4
Consensus / Network Design Space

- **Assumptions**
  - No message loss, FIFO delivery
  - Assumptions
    - Fast Paxos
    - Speculative Paxos / No Paxos
    - Traditional Paxos

- **Programmability**
  - Weak
  - Strong

- **Best effort**
  - Forward packets

- **P4xos (this talk)**
  - Storage and logic
Paxos

Of the various consensus protocols, we focus on Paxos because:

- One of the most widely used
- Often considered the “gold standard”
- Proven correct

“There are two kinds of consensus protocols: those that are Paxos, and those that are incorrect”
— attributed to Butler Lampson
Paxos In the Network

Key questions:

- What parts of Paxos should be accelerated?
- How to map the algorithm to stateful forwarding decisions (i.e., Paxos logic as sequence of match/actions)?
- How do we map from complex protocol to low-level abstractions?
- What are the right interfaces? How do we deploy?
Paxos in a Nutshell

- An execution of Paxos is called an **instance**. Each instance is associated with an ID, called the **instance number**.

- The protocol has two phases. Each phase may contain multiple **rounds**. There is a **round number** to identify the round.
  
  - **Phase 1**: “What instance number are we talking about?”
  
  - **Phase 2**: “What is the value for the instance number?”

- **Observation**: Phase 1 does not depend on a particular value. We should accelerate Phase 2.
Run Phase 1 in a batch, declare the instance numbers to use

Union of all Paxos messages

Paxos Messages

- type
- instance
- round
- vround
- value

Paxos Packets
Paxos In The Switch

Paxos Packets

n m
Paxos In The Switch

When batch fills up, we need to checkpoint
Paxos In The Switch

When batch fills up, we need to checkpoint

Tradeoff with performance and memory
Paxos In The Switch

When batch fills up, we need to checkpoint

Tradeoff with performance and memory

Access dependencies make it hard to implement ring buffer
Paxos In The Switch

When batch fills up, we need to checkpoint.

Tradeoff with performance and memory.

Access dependencies make it hard to implement ring buffer.

Need to use “hacks” to trick the compiler.
Phase 2 Roles and Communication

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 Bottlenecks

Observation: accelerate agreement:
Coordinator and Acceptors
Paxos as Prose

1. (a) If $crnd[c] < i$, then $c$ starts round $i$ by setting $crnd[c]$ to $i$, setting $cval[c]$ to $\textit{none}$, and sending a message to each acceptor $a$ requesting that $a$ participate in round $i$.

(b) If an acceptor $a$ receives a request to participate in round $i$ and $i > rnd[a]$, then $a$ sets $rnd[a]$ to $i$ and sends coordinator $c$ a message containing the round number $i$ and the current values of $vrnd[a]$ and $vval[a]$. If $i \leq rnd[a]$ (so $a$ has begun round $i$ or a higher-numbered round), then $a$ ignores the request.

[Lamport, Distributed Computing ’06]
Paxos as Match-Action

Algorithm 1 shows the pseudocode for the primary leader implementation. The leader receives messages on behalf of proposers. The leader ensures that only one process submits a value to the protocol for a particular instance (thus ensuring that the protocol terminates), and imposes an ordering of messages. When there is a single leader, a monotonically increasing sequence number can be used to order the messages. This sequence number is written to the field of the header. An ordering of messages is maintained by accepting messages only if they are in order (i.e., if the sequence number of the message is greater than the sequence number of any previously accepted message). When a new message is accepted, the sequence number is incremented.

Algorithm 2 shows logic for an acceptor. Acceptors are responsible for choosing a value for a given consensus instance. Learners receive messages in an instance (e.g., because they are forwarding the modified packets. The backup leader must send a unique round number in the Phase 2A messages, which allows the Paxos protocol to tolerate lost or duplicate messages. A majority is equal to the number of acceptors that can be tolerated. A majority is equal to half of the number of acceptors plus one. A majority is equal to the number of acceptors that can be tolerated. A majority is equal to half of the number of acceptors plus one.

Algorithm 3 shows the pseudocode for the learner. Learners provide the interface between the network and the protocol. Learners are responsible for replicating a value. Acceptors must maintain and access the history of proposals for which they have voted. This history ensures that acceptors never vote for different values for the same instance.

Coordinator Algorithm
Paxos as Match-Action

- **Proposer**
  - Encode value in a packet header.
- **Coordinator**
  - If match, add sequence number, and forward
- **Acceptor**
  - If match, compare round field in header, update state, and forward
- **Learner**
  - De-encode and return value to the application.
## Application Interface

<table>
<thead>
<tr>
<th>API Function Names</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>submit</td>
<td>Application to network: Send a value</td>
</tr>
<tr>
<td>deliver</td>
<td>Network to application: Deliver a value</td>
</tr>
<tr>
<td>recover</td>
<td>Application to network: Discover a prior value</td>
</tr>
</tbody>
</table>

C wrapper provides a drop-in replacement for existing Paxos libraries!
P4xos Deployment

Proposer vs. Coordinator

Acceptors vs. Learner/ Application
Experiments

Focus on two questions:

What is the absolute performance?

What is the end-to-end performance?
Absolute Performance

- Measured each role separately on 64x40G ToR switch (Barefoot Tofino) and IXIA XGS12-H as packet sender.
- Throughput is over 2.5 billion consensus messages / second. This is a 10,000x improvement over software.
- Data plane latency is less than 0.1 μs (measured inside the chip).
Application delivers to RocksDB with read and write commands

4.3x throughput improvement over software implementation

73% reduction in latency
Accelerating Execution (Work-in-Progress)

\[ \text{Paxos} \]
\[ \text{Packets} \]
- type
- instance
- round
- vround
- value
- partition

Run multiple Paxi in parallel

\[ \{ \text{Partition application state} \} \]}
Accelerating Execution (Work-in-Progress)

- Not yet done: handling “cross partition” requests
- Must add barriers to synchronize learners
- Fully partitioned workload reaches 500K msgs/sec

RocksDB Throughput vs. Checkpoint Interval
Practical Application: Storage Class Memory

- Fast network interconnect allows users to scale storage and compute separately (i.e., disaggregated storage).

- Several companies, including Western Digital, have developed new types of non-volatile memory:
  - Persistent, with latency comparable to DRAM.
  - But, wears out over time...

- Use in-network consensus to keep replicas consistent.
To Recap

No message loss, FIFO delivery

Best effort

Forward packets

Programmability

NetPaxos

P4xos

“It’s just Paxos!”
To Recap

But, how can we be sure the implementation is correct?
Proving Correctness
(or How Do We Know Our Implementation is Correct?)
An Old Story
You’ve Heard Before

- We checked the Paxos algorithm with SPIN model checker. No problems!
- We wrote the Paxos code.
- We ran in the network, but didn’t get consensus.

There is a bug in our implementation.
Verification is So Tempting…

- To the extent networks are verified, the focus is on forwarding (e.g., no path loops)
- If the network is going to take on more work, how can we be sure that is correct?
- P4 is so tempting to verify: no loops, no pointers, etc.
The specific behavior of a P4 program depends on the control plane.

"If ip.dst is 10.0.0.1, forward out port 1."

We only have half the program!
Hoare Logic

Axioms capture relational properties: what is true before and after a command executes.

$$\vdash \{ P \} \ c \ \{ Q \}$$

If $P$ holds and $c$ executes, then $Q$ holds.

- Standard approach to verification
- Use automated theorem-prover to check if there is an initial state that leads to a violation
- Generate a counter example via weakest pre-condition
P4 + Hoare Logic

Axioms capture relational properties: what is true before and after a command executes.

\[ \vdash \{ P + \text{“control plane assumptions”} \} c \{ Q \} \]

If P plus some assumed knowledge holds and c executes, then Q holds.

- Allow programmers to express symbolic constraints on the control plane in terms of predicates on data plane state
- Combined, the control plane and data plane behave as expected
## Verification Challenges

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4 does not have a formal semantics</td>
<td>We had to define one via translation</td>
</tr>
<tr>
<td>What should the annotations look like?</td>
<td>Leveraged our domain-specific knowledge to define language</td>
</tr>
<tr>
<td>How do we make the solver scale?</td>
<td>Standing on the shoulders of giants, e.g., passivization</td>
</tr>
<tr>
<td></td>
<td>[Flanagan and Saxe, POPL 2001]</td>
</tr>
</tbody>
</table>
Translate P4 to logical formulas

Define a program logic for P4

Annotate to check for properties

Reduce to SMT problem

Desired Property:

“If the tcp.dstPort is 22, then drop the packet.”
P4v : Basic Approach

action forward(p) { … }
table T {
reads {
tcp.dstPort;
eth.type;}
actions {
drop;
forward; }
}

Desired Property:
"If the round number of arriving packet is greater than the stored round number, then drop the packet."

Translate P4 to logical formulas
Define a program logic for P4
Annotate to check for properties
Reduce to SMT problem
@pragma assume valid(paxos) implies local.round <= paxos.rnd
apply(round_table) {
  if (local.round <= paxos.rnd) { apply(acceptor_table) }
}

@pragma assert valid(paxos) implies local.set_drop == 0

Action failed to set the “drop flag” when the arriving round number is greater than the stored round number.
Ran our verifier on a diverse collection of 13 P4 programs

- Conventional forwarding: Router, NAT, Switch
- Source routing: ToR, VPC
- In-network processing: Paxos, LinearRoad

Most finished in 10s of ms; switch.p4 finished in 15 seconds.

Only system to verify switch.p4
Outlook
Summarizing

- System artifact that can achieve orders-of-magnitude improvements in performance
  - Identified techniques for programming within fundamental hardware constraints
- Novel re-interpretation of the Paxos algorithm
  - Hopefully add clarity through a different perspective
- Mechanized proof of correctness of the implementation
A Few Lessons Learned

- What are good candidate applications for network acceleration?
  - “Squint a little bit, and they look like routing”
  - Applications with transient state, rather than persistent
  - Services that are I/O bound
  - Network acceleration helps latency, but throughput is the big win
What’s Next?

- Very exciting time for networking and systems
- Network programmability provides an amazing opportunity to revisit the entire stack
- Redesign systems using an integrated approach, combining databases, networking, distributed systems, and PL
http://www.inf.usi.ch/faculty/soule/