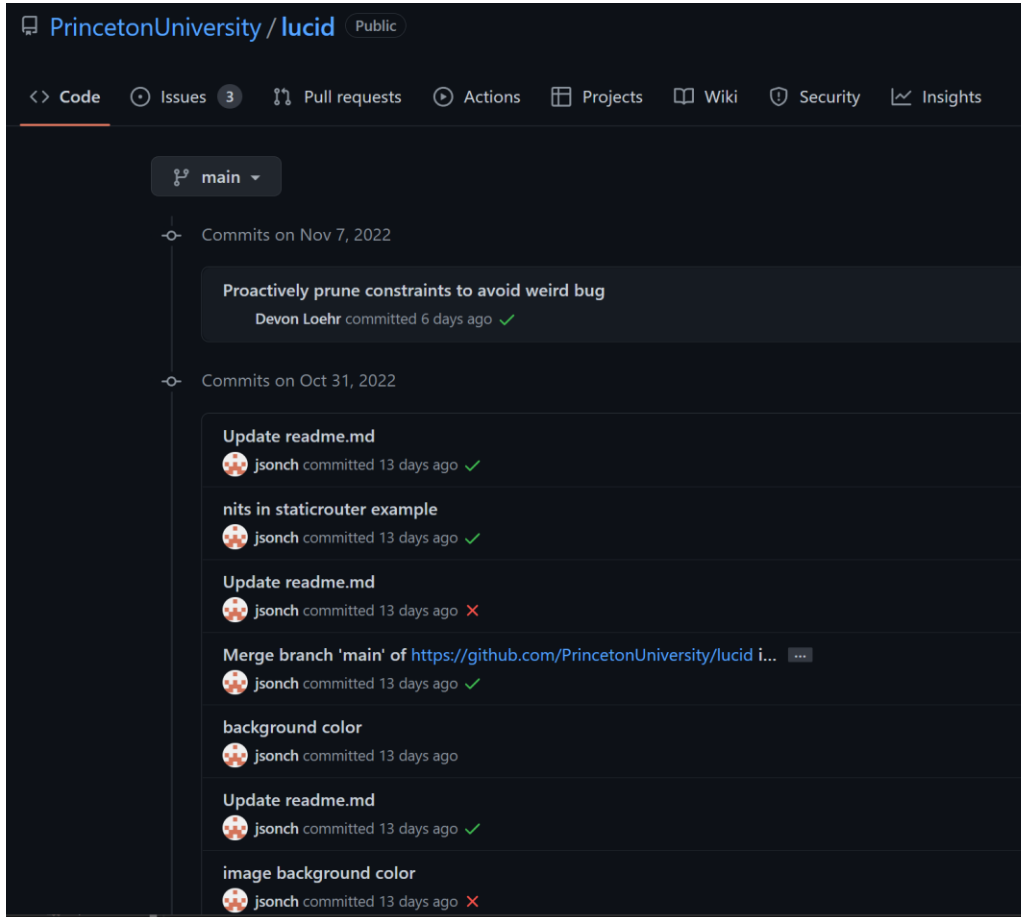
Lucid: A Language for Control in the Data Plane

John Sonchack, Devon Loehr, Jennifer Rexford, David Walker

SIGCOMM '21: Proceedings of the 2021 ACM SIGCOMM 2021 Conference, August 2021

Lucid on GitHub


PrincetonUniversity / lucid Public

Commits on Nov 7, 2022

Proactively prune constraints to avoid weird bug
Devon Loehr committed 6 days ago ✓

Commits on Oct 31, 2022

Update readme.md
jsonch committed 13 days ago ✓

nits in staticrouter example
jsonch committed 13 days ago ✓

Update readme.md
jsonch committed 13 days ago ✗

Merge branch 'main' of https://github.com/PrincetonUniversity/lucid i...
jsonch committed 13 days ago ✓

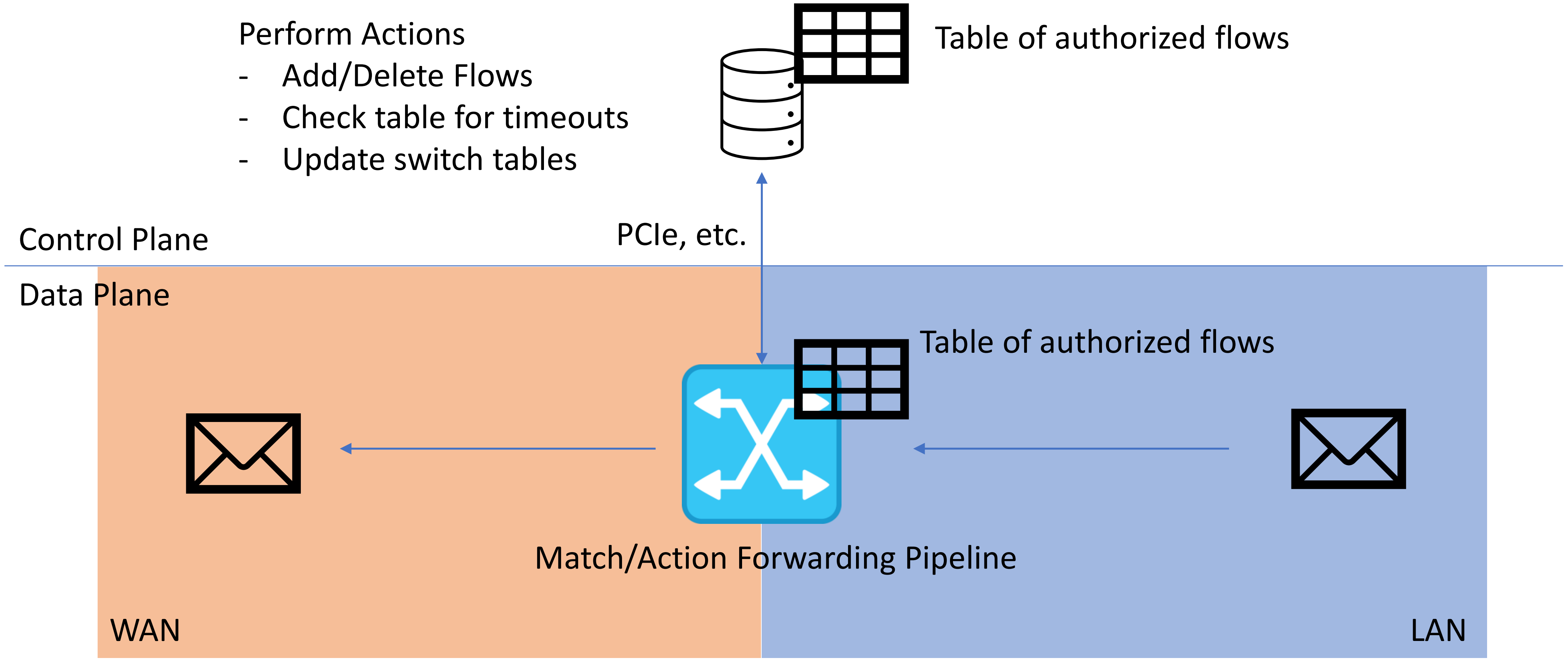
background color
jsonch committed 13 days ago

Update readme.md
jsonch committed 13 days ago ✓

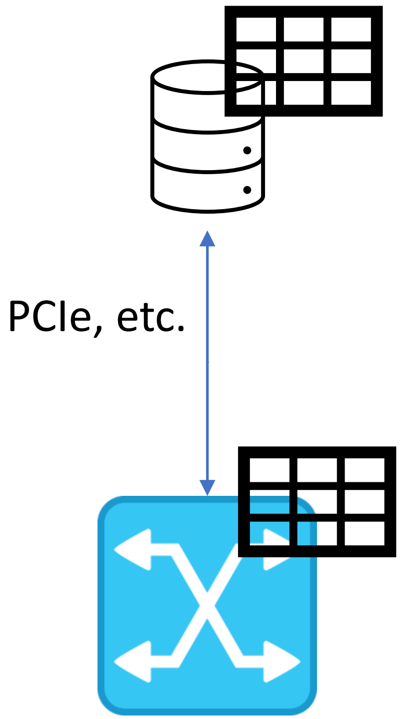
image background color
jsonch committed 13 days ago ✗

- This is a very recent paper and active project.
- Things presented in this paper may not reflect its current status.

Traditional switch architectures

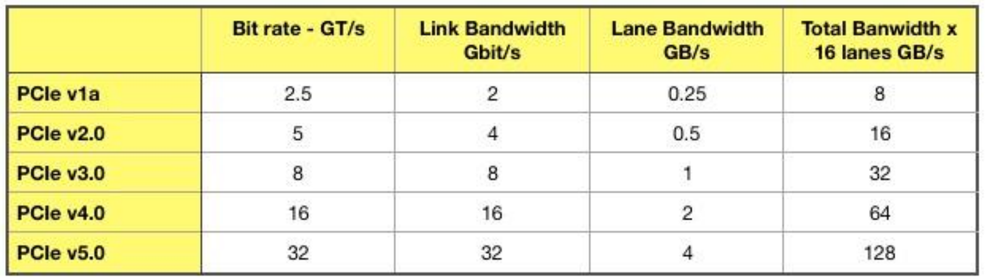

Perform Actions
- Add/Delete Flows
- Check table for timeouts
- Update switch tables
Table of authorized flows
Control Plane
PCIe, etc.
Data Plane
Table of authorized flows
Match/Action Forwarding Pipeline
WAN
LAN

Problems

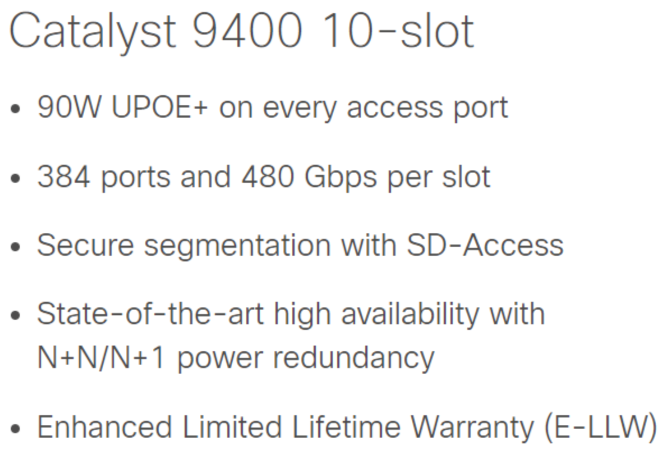


This PCIe link is a bottleneck in packet processing. It introduces latency in table updates.

We used stateful firewalls as an example, but such problem is prevalent in lots of applications.



	Bit rate - GT/s	Link Bandwidth Gbit/s	Lane Bandwidth GB/s	Total Banwidth x 16 lanes GB/s
PCIe v1a	2.5	2	0.25	8
PCIe v2.0	5	4	0.5	16
PCIe v3.0	8	8	1	32
PCIe v4.0	16	16	2	64
PCIe v5.0	32	32	4	128



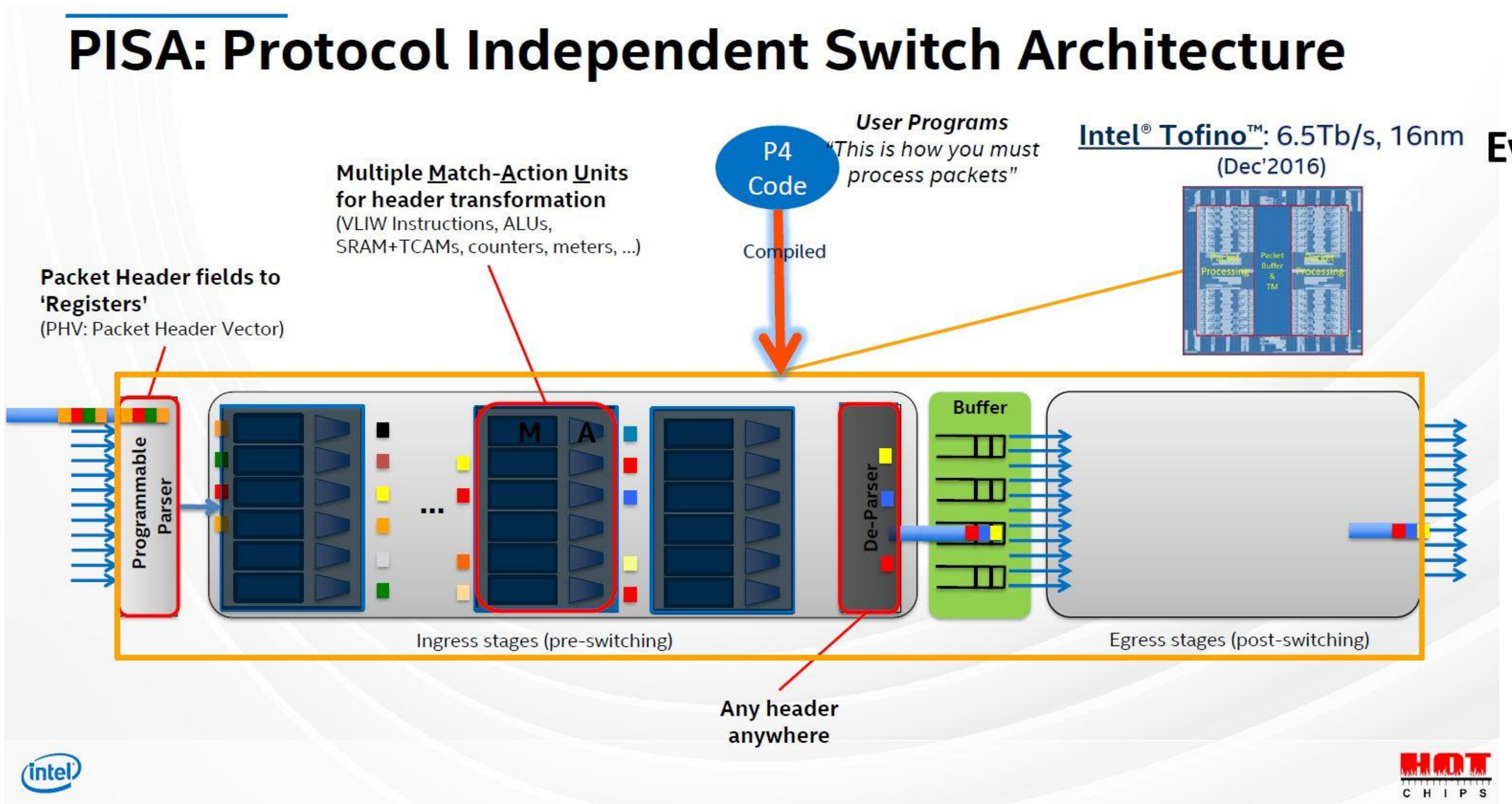
Catalyst 9400 10-slot

- 90W UPOE+ on every access port
- 384 ports and 480 Gbps per slot
- Secure segmentation with SD-Access
- State-of-the-art high availability with N+N/N+1 power redundancy
- Enhanced Limited Lifetime Warranty (E-LLW)

(And up to 9.6Tbps switch fabric!)

PISA Switches

- PISA – Protocol Independent Switch Architecture.
- P4 is a language for programming PISA switches.


PISA: Protocol Independent Switch Architecture
User Programs
"This is how you must process packets"
P4 Code
Compiled
Intel® Tofino™: 6.5Tb/s, 16nm
(Dec'2016)
Packet Processing
Packet Buffer & TM
Packet Processing
Multiple Match-Action Units for header transformation
(VLIW Instructions, ALUs, SRAM+TCAMs, counters, meters, ...)
Packet Header fields to 'Registers'
(PHV: Packet Header Vector)
Programmable Parser
M
A
...
De-Parser
Buffer
Ingress stages (pre-switching)
Egress stages (post-switching)
Any header anywhere
intel
HOT CHIPS

Evidence: Tofino 6.5Tb/s switch (arrived Dec 2016)

The world's fastest and most programmable switch.
No power, cost, or power penalty compared to fixed-function switches.
An incarnation of **PISA (Protocol Independent Switch Architecture)**

PISA Power

- Flexible parsing and manipulation of packets
- Can update persistent states (register arrays)
- …

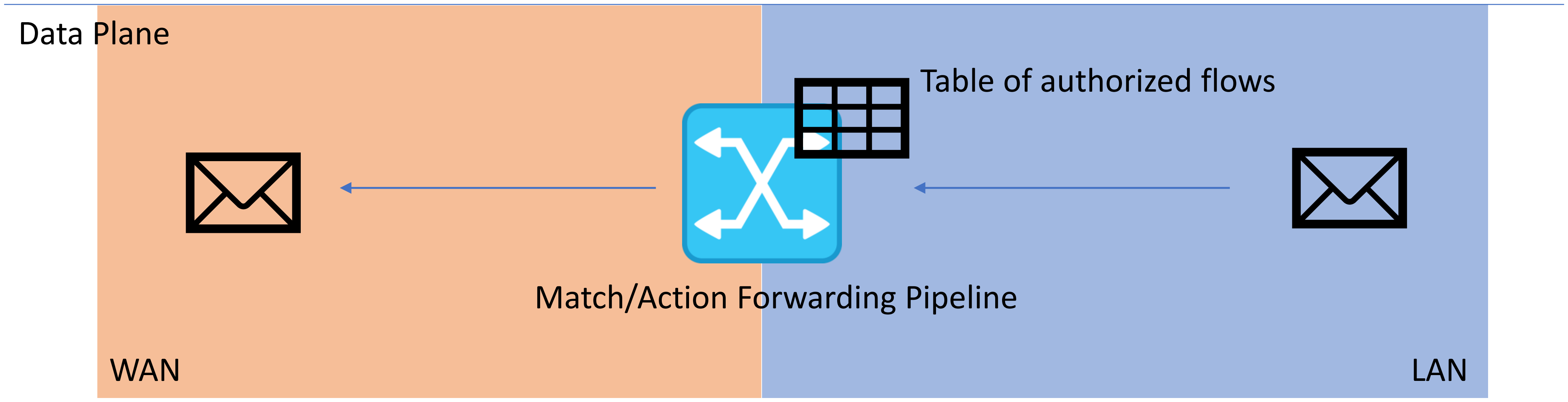
- We can now implement some control logic directly in the dataplane.

PISA Stateful Firewall

On LAN packet ingress
- Store the flow in SRAM/TCAM on switch.

On WAN packet ingress
- Check SRAM/TCAM for corresponding flow.

Periodical packet generator to trigger timeout detection.



New Problem

- This programming model is weird!
- Deal with the following concepts on P4:
  - Match tables, Actions
  - Packet parser
  - Register Arrays
  - Register Actions
  - Packet generator
  - Packet replication
  - Packet recirculation
  - Priority-Based Flow Control queues

New Problem

- As a higher-level language, the primitives and programming model provided by P4 is tedious to work with.
    - Express control logic in the language of packet processing logic, which consists of disjoint components such as parser, MAU and recirculation.
    - Orchestrate other components (such as packet generators) for more sophisticated controls, such as state management.
- Example: Implementing loops in P4 programs for applications such as fast rerouter.

Case Study: Fast Rerouter

- Periodically send packets that ping all neighbors.
- Update link statuses.
- Decide which link to go on egress.

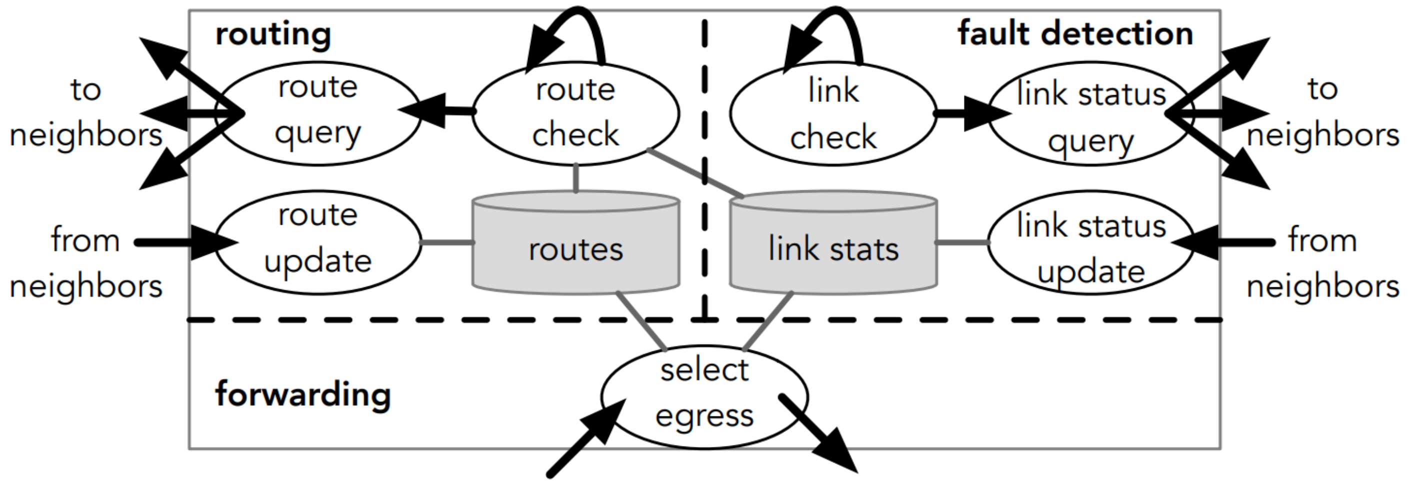


Figure 2: fast rerouter architecture. Circles represent operations, with arrows for (possibly delayed) control flow. Shaded objects are data structures.

Problem 1: Necessary Loops

- P4 is designed that all operations on the dataplane go at line rate, and operations are expressed in MAU in pipelines.
- No loops: The PISA architecture cannot loop in a single stage.

- However in the fast rerouter, we need a loop that iterates through all links and send pinging packets.
  - Generally, loops are needed for all maintenance tasks.

Problem 1: Necessary Loops

- Solution: Use the recirculation path.

- Design a packet with headers that represent this operation. Include a counter in the header.
- Recirculate this packet with the recirculation path, ping a neighbor according to the counter when this packet is met.
- Discard the packet after the counter meets the total number of links.

Problem 2: Scheduling

- We do not want this pinging to happen all the time.
- Use a packet generator to generate the aforementioned packet.

- We want to tell our link to other switches.
- Send the packet to someone else.

Problem 3: Unexposed Constraints

```
X = RegisterArray<32>(128);
Y = RegisterArray<32>(128);

ingress() {
  if(foo) {
    tmp = X[0];
    Y[0] = tmp;
  else {
    tmp = Y[0];
    X[0] = tmp;
  }
}
```

- This code will not compile to P4.
- P4 programs store persistent state in RegisterArrays, and use RegisterActions with arbitrary blocks of C-like code to operate on those arrays.
- However, not all RegisterActions can be actually implemented by Tofino.

Problem 3: Unexposed Constraints

```
X = RegisterArray<32>(128);
Y = RegisterArray<32>(128);

ingress() {
  if(foo) {
    tmp = X[0];
    Y[0] = tmp;
  else {
    tmp = Y[0];
    X[0] = tmp;
  }
}
```

- This code will not compile to P4.
- Registers are associated with a single stage.
- The first condition means X[0] must appear before Y in the pipeline.
- The second conditions means Y[0] must appear before X[0] in the pipeline.
- You'll get a cryptic message: *Table placement cannot make any more progress.* Does not tell you what's actually wrong.

P4 Complexity

- From the discussion we can see that implementing a fast rerouter in P4 involved a lot of low-level programming.
- In some cases, abstraction is broken: The underlying constraints gets propagated to the language level, and the developer is forced break the abstraction and inspect the details.

- Enter Lucid, a high-level abstraction that simplifies this process.

Lucid

- A high-level language that raises the level of abstraction and make programming easier.

- Designed for Tofino, compiles to P4.

Abstractions

P4

- Match tables
- Actions
- Packet parser
- Register Arrays
- Register Actions
- Packet generator
- Packet replication
- Packet recirculation
- Priority-Based Flow Control queues

Lucid

- Events
- Handlers
- Arrays

Abstraction: Events

- Events can be both control operations and data packets.
    - Also an abstraction of application-layer messages, such as the pings.
    - Events can be sent to other switches, or a switch itself.
- Four members: Name, data, time and place.
- Constructs a “multithreading” programming model, since events can be handled in parallel.

Abstraction: Handlers

- All computation happens in handlers. Specifies what happens when the switch receives an event.
- Compiles to multiple primitives: Tables, ALUs, stateful ALUs.
- Executes in a single-pass through the pipeline.

Abstraction: Arrays

- Store persistent states.
- Provides a functional interface for operations that can be done with a single pass of ALU.

- Helps writing modular and reuseable code.

Fast-Rerouter Events & Handler

```
global pathlens = new Array<<32>>(table_size);
memop incr(int stored, int added) {
  return stored+added;
}
fun int get_pathlen(int dst) {
  return Array.get(pathlens, dst, incr, x);
}
```

Returns incr(pathlens[dst], x)

Fast-Rerouter Events & Handler

```
event route_query(int sender_id, int dst);
event route_reply(int sender_it, int dst, int
pathlen);
event check_route(int dst);

# Replies to a route query from sender_id
handle route_query(int sender_id, int dst) {
  int pathlen = get_pathlen(dst);
  event reply = route_reply(SELF, dst, pathlen);
  generate Event.locate(reply, sender_id);
}
```

Schedules a reply event to run on sender_id. Lucid’s way of sending response.
route_reply function on the sender will handle this event and update registers.

Event Generation

- There are two event compositors in Lucid:
  - `Event.locate()` schedules an event on some switch.
  - `Event.delay()` schedules an event to delay until being handled.
  - Two combinators can be used together.

- Allows a control operation to be broken into a series of events.
- Allows scheduling events after some time or on other switches.
  - Loops can be easily implemented by starting another event at the end of handler.
  - Allows easier implementation of distributed systems.

Event Dispatching

- Local, immediate events:
  - The event is “processable.” It is sent into the pipeline, and the match-action tables are executed.
- Local, non-immediate events:
  - The packet stays in a special delay queue at recirculation egress until time is up.
  - The delay queue is paused most of the time, and gets unpaused periodically.
- Remote events:
  - The event packet gets sent to a port or a multicast group.

Memops

```
memop incr(int stored, int added) {
  return stored+added;
}
```

- A special type of function that carries some simple operation to be completed in one ALU.
- Compiler will throw an error if the operation is too complex.
  - Either a simple return statement, or an if statement with a return statement in each branch.
  - Each variable used at most once per expression.
  - ALU-supported operators only
- Prevents developers unknowingly writing unimplementable complex operations.

Ordered Data Access

```
X = RegisterArray<32>(128);
Y = RegisterArray<32>(128);

ingress() {
  if(foo) {
    tmp = X[0];
    Y[0] = tmp;
  else {
    tmp = Y[0];
    X[0] = tmp;
  }
}
```

- Constraint: Persistent data must be partitioned across the stages of a feed-forward pipeline. Operations that happens in later stages cannot read/modify data in previous stages.
- Lucid treat declarations as high-level intentions of how the developer want the data to be placed.

Ordered Data Access

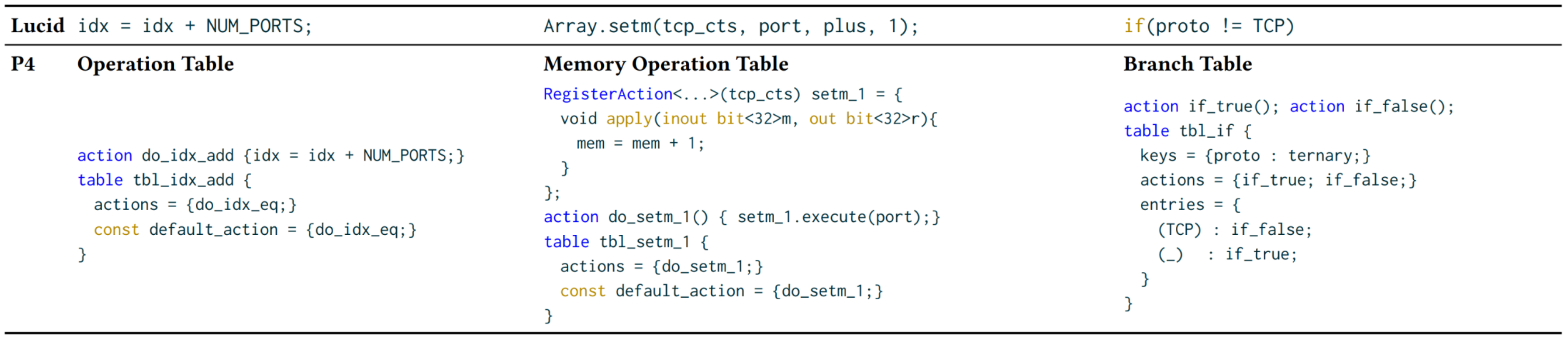
- A program is well-ordered if all data access follows the declaration order.
- This is generalizable to any PISA switch, since the constraint is universal.
- Lucid will throw a source-level error if the order is violated.

Compilation & Optimizations

- Lucid is compiled to P4_16 for Tofino.
  - Events map to packet headers.
  - Event scheduler is static code (library)
  - Handlers are translated to atomic P4 tables, and control flow is optimized.

Handler Compilation

- The handler body is broken into a graph of execution. Each statement is implementable in a single ALU.
- Three types of tables



Lucid	`idx = idx + NUM_PORTS;`	`Array.setm(tcp_cts, port, plus, 1);`	`if(proto != TCP)`
P4	**Operation Table**	**Memory Operation Table**	**Branch Table**

```
action do_idx_add {idx = idx + NUM_PORTS;}
table tbl_idx_add {
  actions = {do_idx_eq;}
  const default_action = {do_idx_eq;}
}
```

```
RegisterAction<...>(tcp_cts) setm_1 = {
  void apply(inout bit<32>m, out bit<32>r){
    mem = mem + 1;
  }
};
action do_setm_1() { setm_1.execute(port);}
table tbl_setm_1 {
  actions = {do_setm_1;}
  const default_action = {do_setm_1;}
}
```

```
action if_true(); action if_false();
table tbl_if {
  keys = {proto : ternary;}
  actions = {if_true; if_false;}
  entries = {
    (TCP) : if_false;
    (_)   : if_true;
  }
}
```

Figure 7: Examples from Figure 6 of the three kinds of Atomic P4 tables that the Lucid compiler generates.

Optimizing Control Flow

- Inlining branch operations (1->2)
  - Eliminate branch tables by placing the conditions in non-branch tables.
- Rearranging tables(2->3)
- Merging tables and actions.

```
Array nexthops = new Array<<32>>(NUM_HOSTS);
Array pcts = new Array<<32>>(NUM_PORTS_X3);
Array hcts = new Array<<32>>(NUM_HOSTS);
memop plus(int cur, int x){return cur + x;}

event count_pkt(int dst, int proto);
handle count_pkt (int dst, int proto) {
    int idx = Array.get(nexthops, dst);
    if (proto != TCP) {
        if (proto == UDP)
            idx = idx + NUM_PORTS;
        else
            idx = idx + NUM_PORTS_X2;
    }
    Array.set(pcts, idx, plus, 1);
    if (proto == TCP)
        Array.set(hcts, dst, plus, 1);
}
```

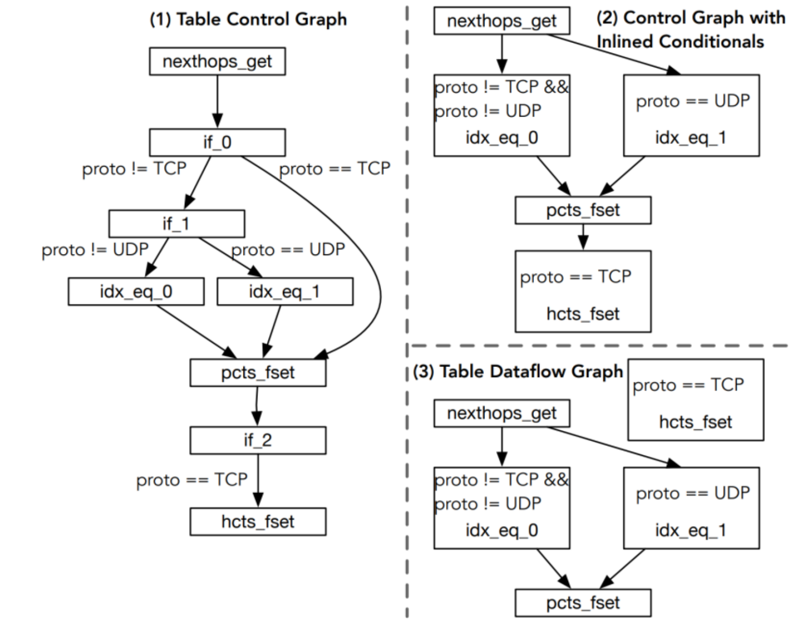
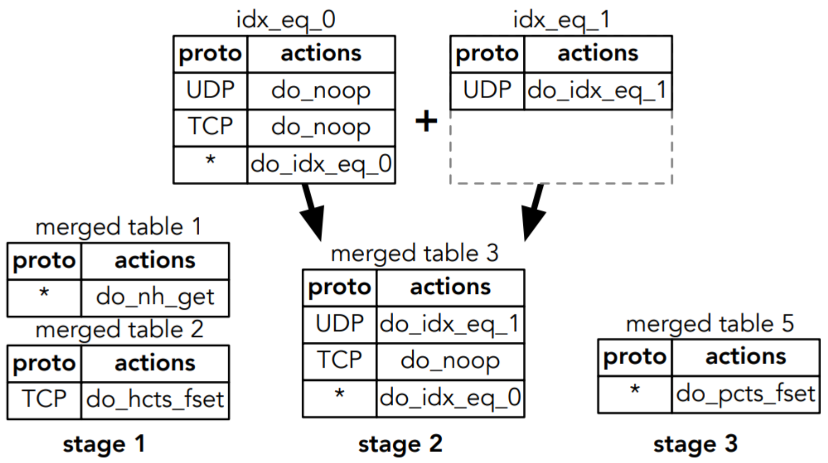


Figure 6: Top: a Lucid handler using only atomic statements. Bottom: the handler represented as an atomic table graph (1) and optimized to require fewer pipeline stages (2 and 3).



idx_eq_0

proto	actions
UDP	do_noop
TCP	do_noop
*	do_idx_eq_0

+

idx_eq_1

proto	actions
UDP	do_idx_eq_1

merged table 1

proto	actions
*	do_nh_get

merged table 2

proto	actions
TCP	do_hcts_fset

stage 1

merged table 3

proto	actions
UDP	do_idx_eq_1
TCP	do_noop
*	do_idx_eq_0

stage 2

merged table 5

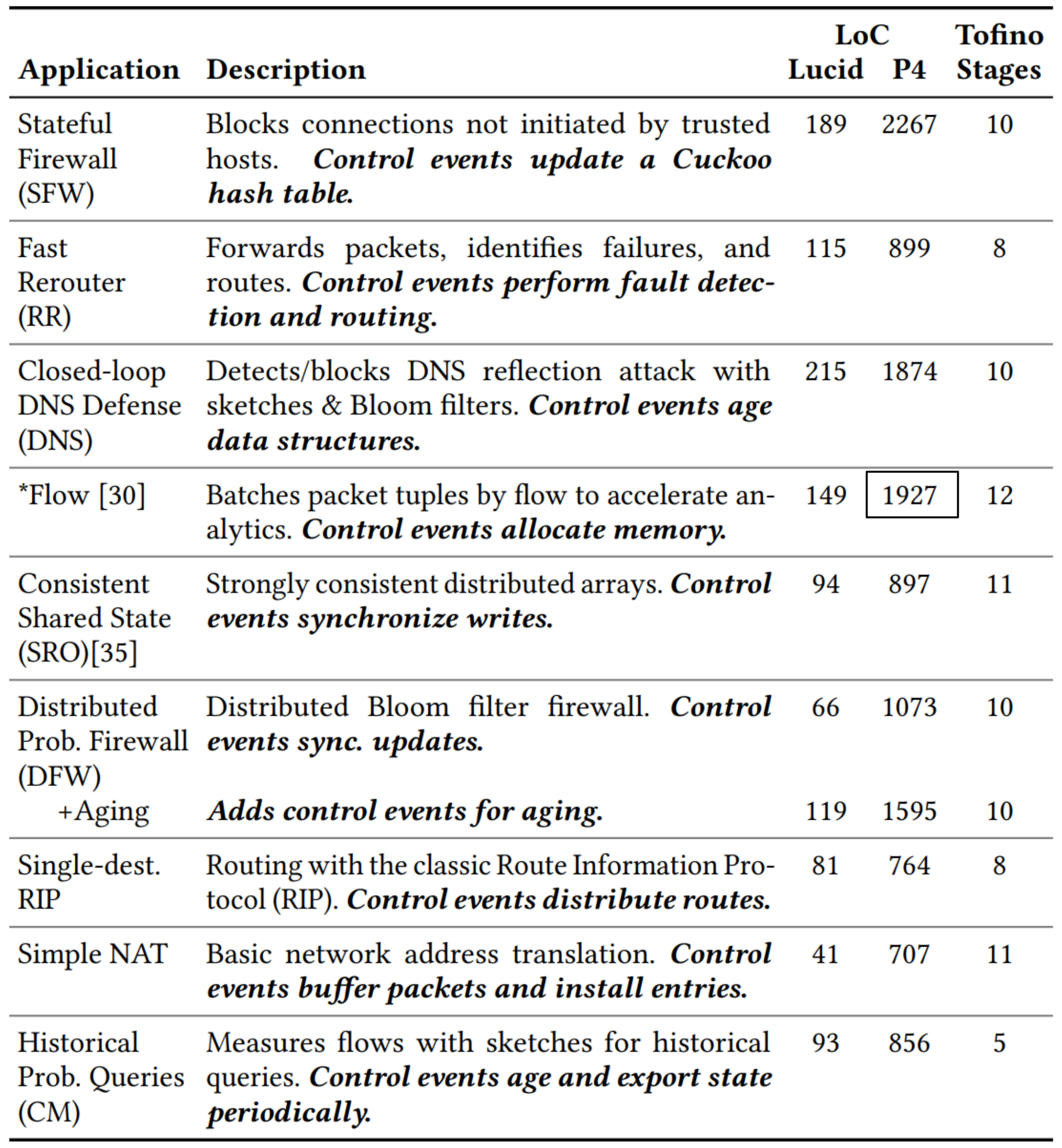
proto	actions
*	do_pcts_fset

stage 3

Figure 8: Merged tables for the program in Figure 6.

Evaluation: LoC

- Estimated 10 ~ 15% of P4 code length, based on *Flow.
- Shown to the right: Lucid program lines & generated P4 lines.
- A PhD student new to P4 wrote a non-trivial Lucid program in less than an hour.



		LoC		Tofino
Application	Description	Lucid	P4	Stages
Stateful Firewall (SFW)	Blocks connections not initiated by trusted hosts. ***Control events update a Cuckoo hash table.***	189	2267	10
Fast Rerouter (RR)	Forwards packets, identifies failures, and routes. ***Control events perform fault detection and routing.***	115	899	8
Closed-loop DNS Defense (DNS)	Detects/blocks DNS reflection attack with sketches & Bloom filters. ***Control events age data structures.***	215	1874	10
*Flow [30]	Batches packet tuples by flow to accelerate analytics. ***Control events allocate memory.***	149	$\boxed{1927}$	12
Consistent Shared State (SRO)[35]	Strongly consistent distributed arrays. ***Control events synchronize writes.***	94	897	11
Distributed Prob. Firewall (DFW)	Distributed Bloom filter firewall. ***Control events sync. updates.***	66	1073	10
+Aging	***Adds control events for aging.***	119	1595	10
Single-dest. RIP	Routing with the classic Route Information Protocol (RIP). ***Control events distribute routes.***	81	764	8
Simple NAT	Basic network address translation. ***Control events buffer packets and install entries.***	41	707	11
Historical Prob. Queries (CM)	Measures flows with sketches for historical queries. ***Control events age and export state periodically.***	93	856	5

Figure 9: Applications with data plane-integrated control, implemented in Lucid and compiled to the Barefoot Tofino. The role of control events is bolded.

Evaluation: Optimization effectiveness

- Greater benefit to complex applications.
- High ALU instruction per stage, good parallelism exploitation.

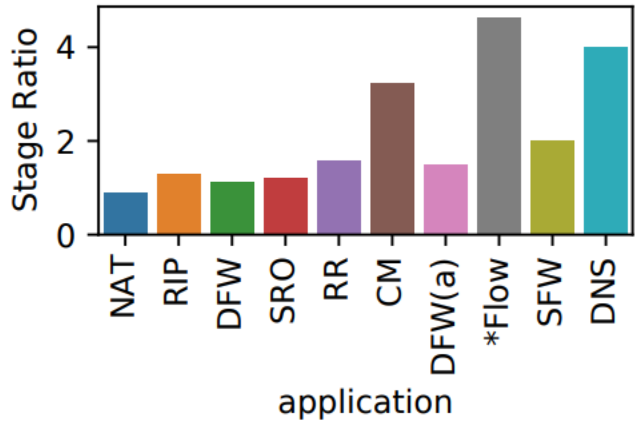


Figure 12: Optimized stage count vs unoptimized.

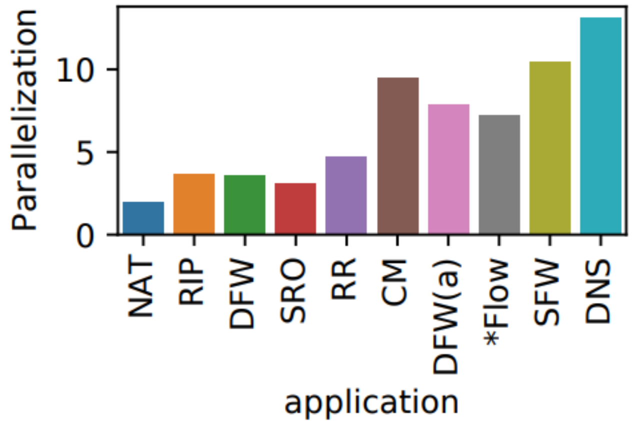


Figure 13: ALU instrs. per stage in optimized code.

Evaluation: Event Scheduler

- The effectiveness of using pausable queues in the event scheduler.
- Increased buffer utilization and error in timing.

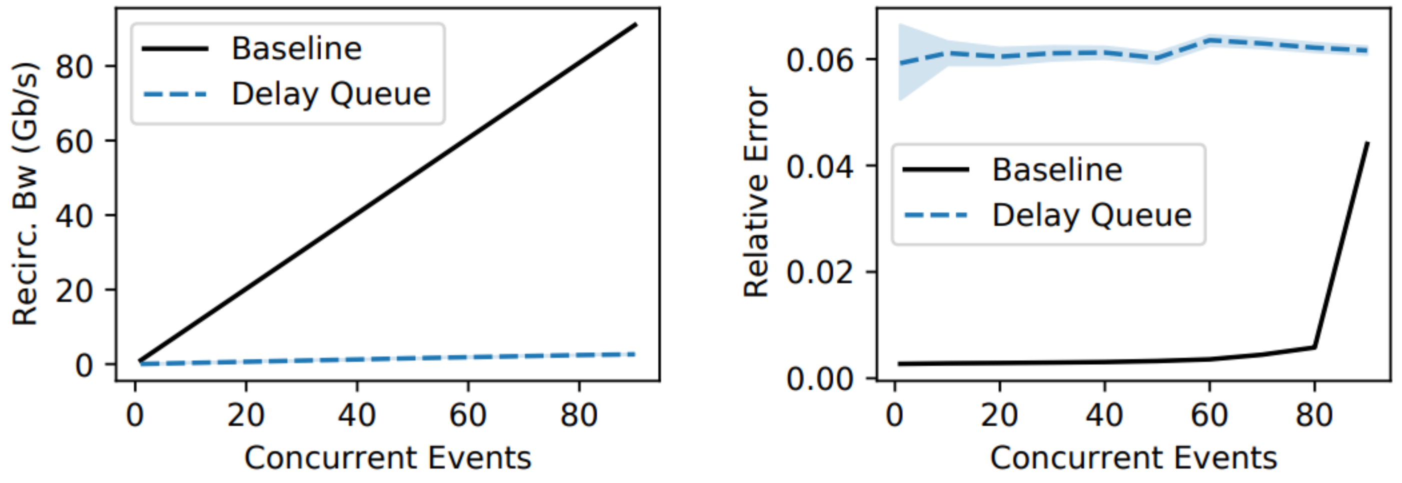


Figure 14: Pausable queue overhead and accuracy.

Evaluation: Stateful firewall

- Flow installation time is 300-400x times better.

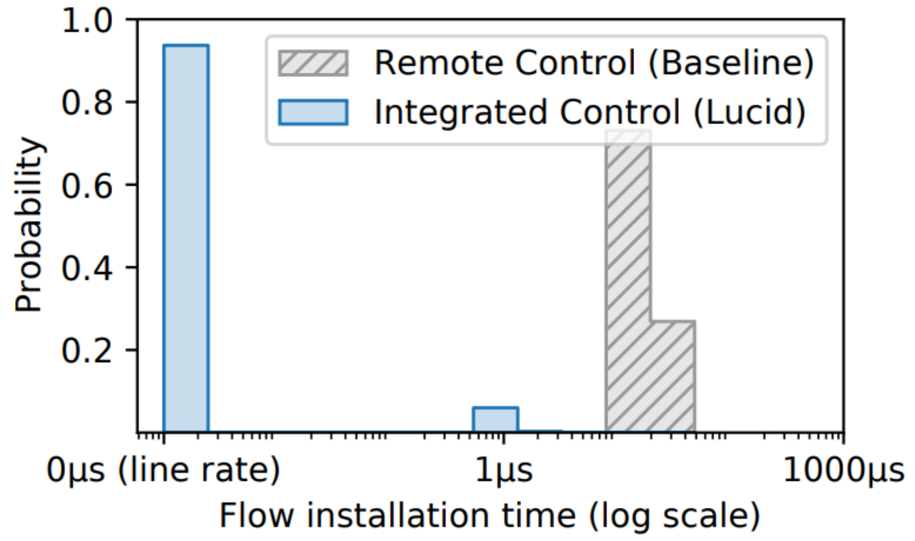


Figure 17: Stateful firewall flow installation times. 1000 trials using a 2048-element table with a load factor of $.3125$**.**