P4 GSoC Kickoff Meeting

June 11, 2025

Google
Summer of Code

Agenda

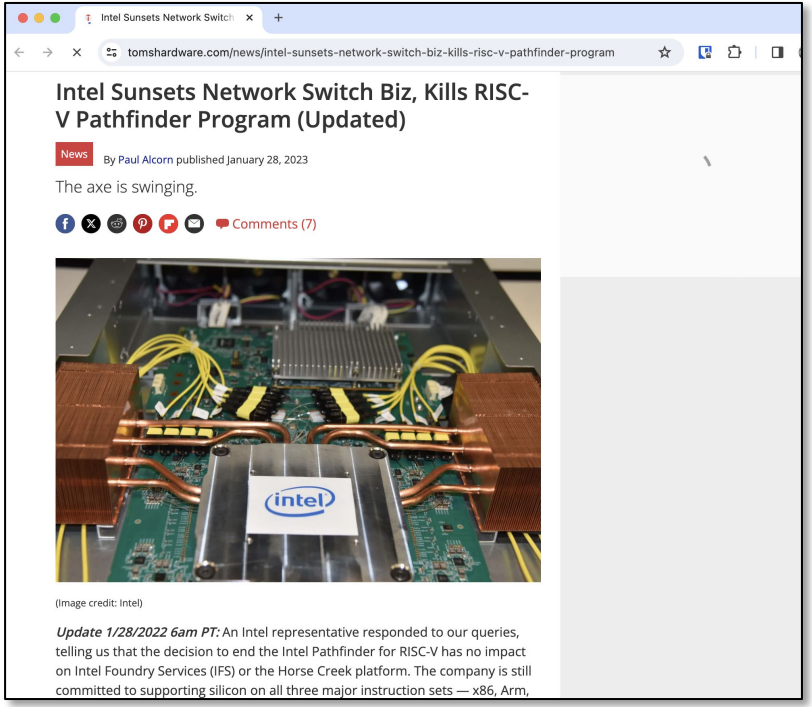
- Introduction Talk by Nate Foster
- P4 GSoC Project Proposals
  - Xiyu Hao – BMv2 with all possible output packets
  - Vineet Goel – P4Simulator: Enabling P4 Simulations in ns-3
  - Xiaomin Liu – P4MLIR: MLIR-based high-level IR for P4 compilers
  - Advay Singh – Gigaflow: A Smart Cache for a SmartNIC
  - Sankalp Jha – SpliDT: Scaling Stateful Decision Tree Algorithms in P4

- To learn more about P4 GSoC and stay updated:
  - https://github.com/p4lang/gsoc
  - https://p4lang.zulipchat.com/ → #gsoc

The Road Ahead

Nate Foster

In 2023, Intel announced the end of Tofino of P4-programmable switches...



The Cynical View

"Programmability never really made sense in the network fabric"

"The open-source networking era is officially over"

"The P4 language will die without realistic hardware targets"

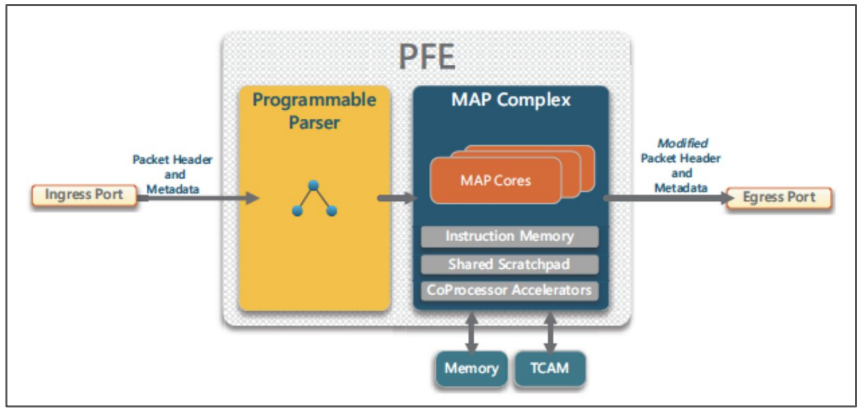
My Own View

- Network programmability is far from over
  - In fact, it's an *inevitable* trend
  - Vendors have long known how to build programmable ASICs
  - They just didn't expose it to end users like Tofino does
- Moore's Law is coming for the network fabric too
  - With Terabit link speeds, in-network processing will be a necessity
  - Vendors will want to differentiate, offering different network services
  - Programmability and open-source are the only sane ways to support rich customization
- P4 has a promising future
  - Being quietly used by multiple hardware vendors
  - Growing use of the language on software targets and for validation

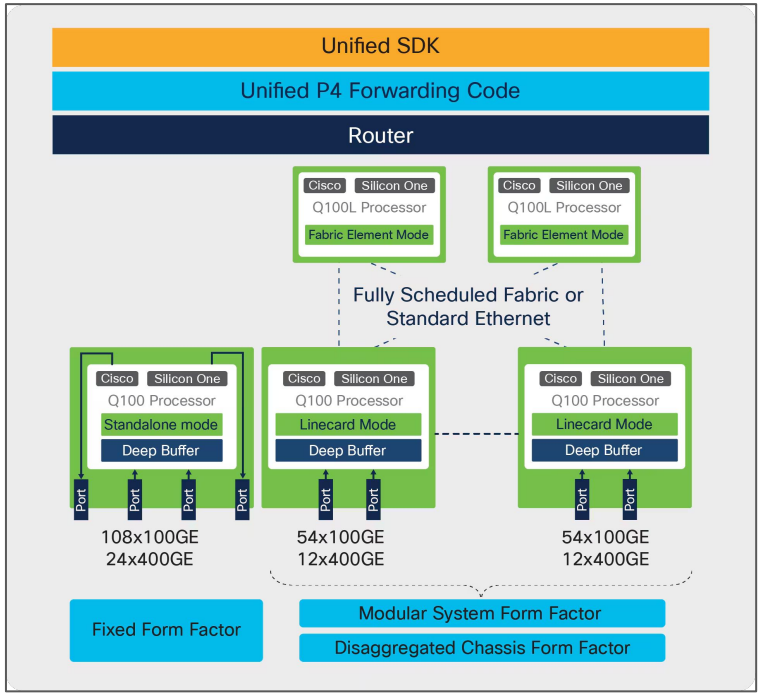
The New P4 Language Consortium

- Since 2024, the P4 project has been operating under the Linux Foundation's industry-standard model for open-source
- Fiscal endowment from ONF's reserves plus a contribution from LF
- The Governing Board has recruited a set of dedicated members, including hardware vendors and users of P4!

P4 on Switches



XSight X2



Cisco SiliconOne

P4 on NICs

- Leading vendors have announced P4 support on SmartNICs
  - AMD (Pensando DPU, Alveo FPGAs)
  - Intel (E2000 IPU)
  - NVIDIA (Bluefield DPU)
- NIC-based use cases are helping drive evolution of the P4 language
  - Features for transport, cryptography, etc.
  - Interfaces with memories and interconnects

P4 in Software

- While P4 was designed as a hardware DSL, there is also growing interest in efficient software pipelines
- P4-DPDK has become the de-facto reference implementation of the language, replacing the Bmv2 software switch
- A group of kernel hackers have added support for P4 to Linux, in the TC subsystem
- These efforts have led to a noticeable uptick in contributions to open-source software for P4 (e.g., the p4c compiler)

P4 for Validation

- One of the most surprising (to me!) uses of P4 has been as a specification language for fixed-function network devices
- Google's Switch-V project uses P4 to model and validate their data center switches [SIGCOMM '22]
- Microsoft's DASH project (part of SoNIC) is using P4 to specify the functionality of reference pipelines for NICs
- `p4testgen` [SIGCOMM '23] provides automatic test generation for P4, and is in use at several companies

Wrapping Up...

- 2025 will continue to be a time of transition for P4...
- But I am optimistic...
- I believe that network programmability is inevitable
- P4 has a solid foundation (funding, governance), a growing set of hardware targets, and a diverse set of use cases that is growing organically in academia and industry
- We are incredibly grateful for your contributions to the language ecosystem through GSoC!

P4 GSoC Project Proposals

BMv2 with all possible output packets

Xiyu Hao

Xiyu Hao

Master’s in Computer Science at New York University

Interests:

- Distributed Systems - BFT Problem
- Networks
- Compilers

BMv2 with All Possible Output Packets

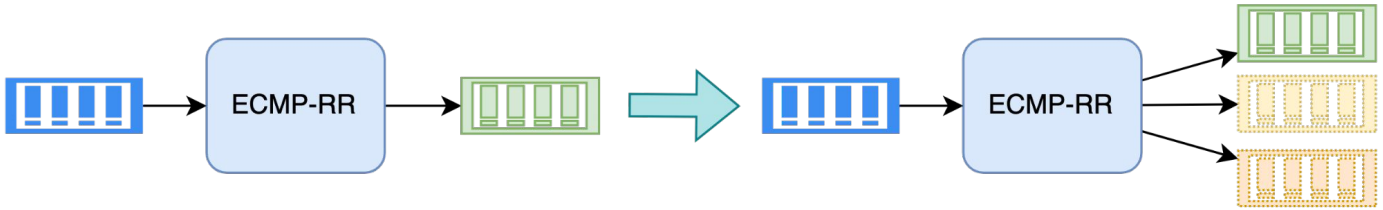
Keywords:

1. BMv2: Behavioral Model V2. A software simulator for P4 Dataplane & Controlplane.
   Useful for quick P4 program and model (arch) development.
2. All possible outputs: as an example, P4 supports constructs that can be used to implement multi-path behaviors. “All possible outputs” refers to the outcomes of taking all such paths.

BMv2 with All Possible Output Packets

Example: ECMP - Round Robin

- There exists a “selector” in the ECMP-RR construct that selects between 3 egress ports in a round-robin fashion.
- Traditionally, one header state is obtained for selecting one egress port.
- Goal: to obtain all three header states, and have them continued in the pipeline.



BMv2 with All Possible Output Packets

Why Does This Matter?

- Diff Testing: Compare with the result of deployed P4 switch. Helpful for fast program and selector (hash function) behavior validation.
- Deterministic behavior and better test coverage:
  Constructing proper inputs to explore code paths produced by the multi-path constructs is a burden for test writers.
  With this feature, there would be much fewer tests need to be built.

BMv2 with All Possible Output Packets

Challenges:

- How to “continue” the replicated headers to flow through the pipeline in a preferably non-hacky way.
- How to preserve the switch and program states (*e.g.* counters and metadata) for each replicated packet.

BMv2 with All Possible Output Packets

Approach:

- In order to continue the header in pipeline, we record the next control flow node (table) after the selected action for each replicated headers. Then continue from the node one header by one header.
- Implement a desirable “snapshot” mechanism that captures the states to preserve,

Deliverable:

- An interface/mode/pragma that replicates header states for all possible selections and obtains the resulting states.

BMv2 with All Possible Output Packets

Q & A

P4Simulator: Enabling P4 Simulations in ns-3

Vineet Goel

About Myself

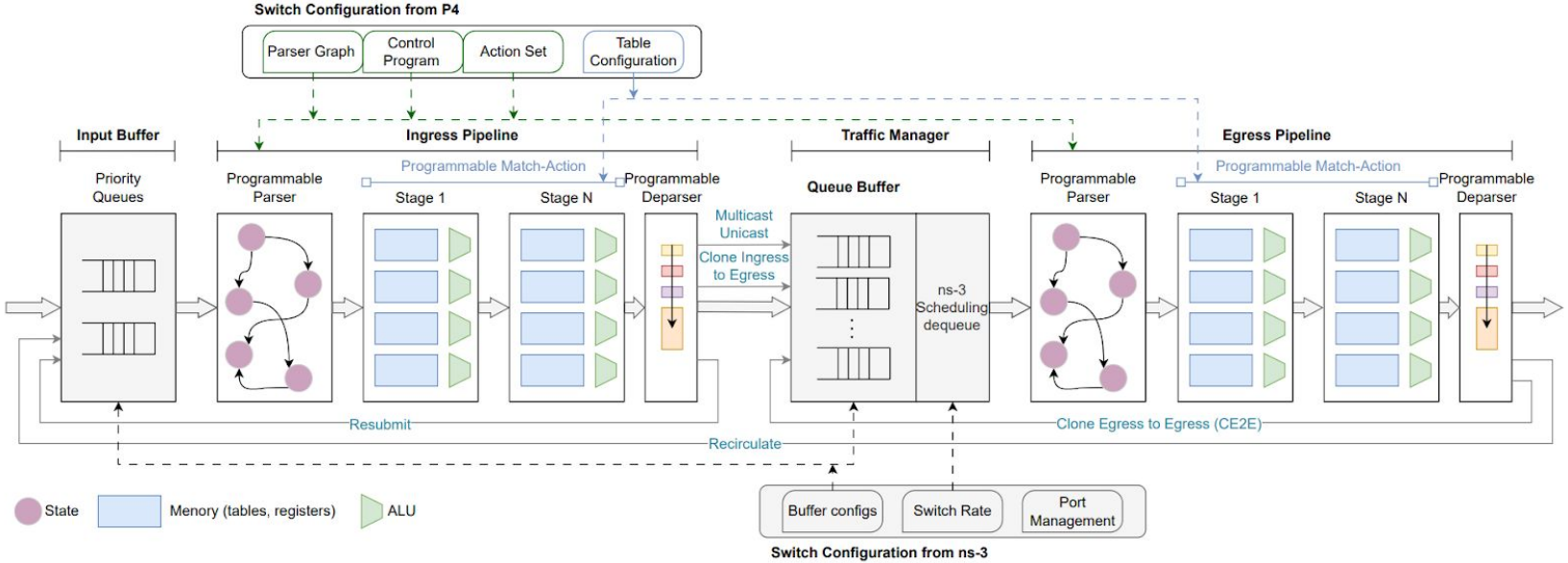
- Vineet Goel
- Undergrad student at IIT Roorkee
- Interested in Web3, system design and full stack development
- Github :Vineet1101
- Linkedin : Vineet
- Email Id: Vineetgoel692@gmail.com
- Hobbies: Playing video games

Introduction to P4Sim

P4sim is a high-performance P4-driven simulation framework built on bmv2 and NS4, seamlessly integrated with ns-3.

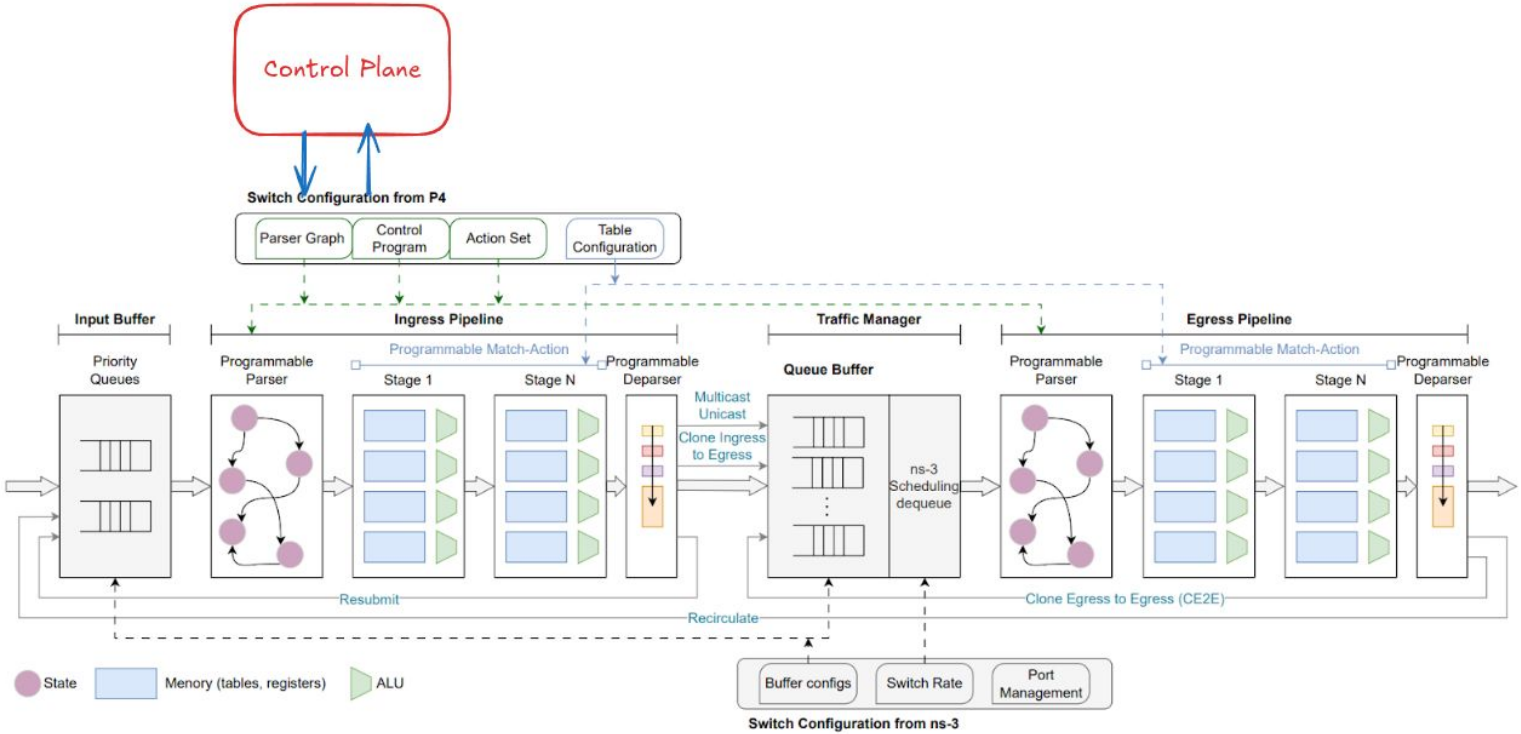
- **Efficient Switch Modelling**
- **Modern P4 Architecture Support**
- **Protocol Independence**
- **Full ns-3 Integration**
- Github Repo:P4Sim

The Structure of the P4Sim



The P4 switch model in ns-3, based on PSA architecture, includes a P4-configured packet processing pipeline and ns-3-managed buffer scheduling for virtual time simulation.

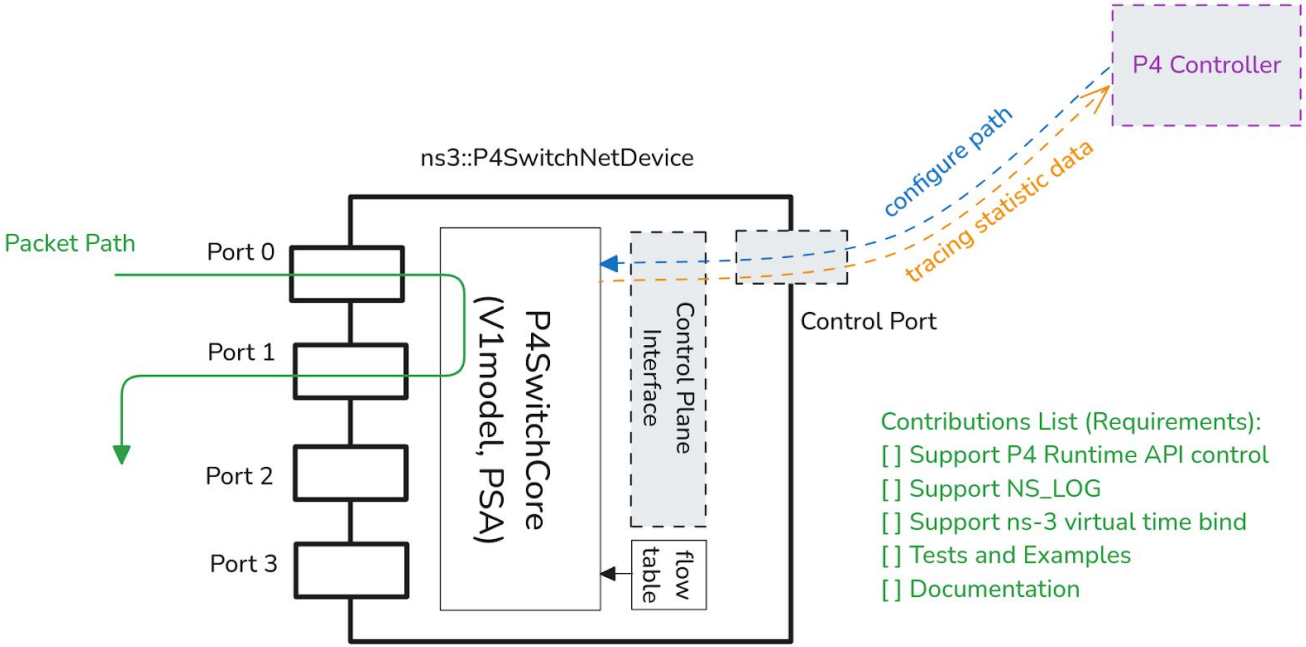
Limitation of Current P4Sim



GSOC: Project Goals

- Control Plane Implementation for P4Sim
- Data Collection and Tracing Mechanism
- Testing and Example Scenarios
- Documentation

Technical Approach to the Project



System design integrates P4 runtime control in ns-3; the data plane is implemented, but the control plane is still missing.

Timeline of the Project

- 2 June- Coding Period for GSOC starts.
- 15 July- Mid Term Evaluation
  - Ensure a working control plane setup
- 1 September- Final Evaluation
  - Delivering final code, tests, and documentation

Benefits to the Community

- Enablement of centralized control in P4 simulator
- Educational Utility for P4
- Improved P4 program debugging with ns3 virtual time
- Offers broader support for P4

Thank you everyone

Q/A

P4MLIR: MLIR-based high-level IR for P4 compilers

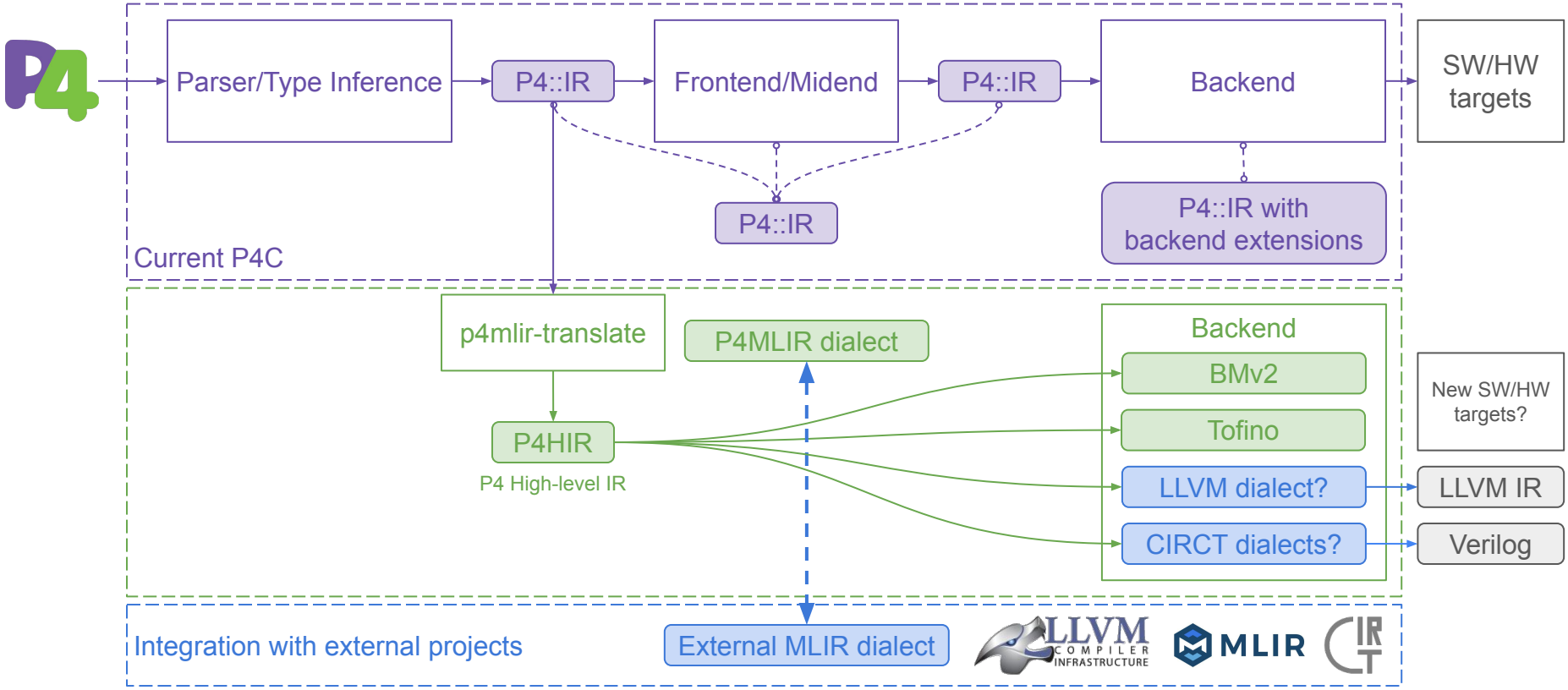
Xiaomin Liu

Introduction

- Xiaomin Liu (SHOW-min LIH-oo)
- From Queens, NYC
- Bachelor’s Degree in Computer Science at NYU
- Interested in compilers & programming languages
- Interning at Google this fall working on their SmartNICs

Motivation

- MLIR (Multi-Level Intermediate Representation) is a compiler framework for representing code at various levels of abstractions using custom IR called dialects
- P4C currently suffers from performance issues rooted in its design:
  - Excessive memory usage – for example def-use pass can consume 16GB+
  - Has immutable IR but modifies its IR, which causes inefficient copying of nodes
  - Long compile times compared to other compilers even though p4 is a simple language


Parser/Type Inference
P4::IR
Frontend/Midend
P4::IR
Backend
SW/HW targets
P4::IR
P4::IR with backend extensions
Current P4C
p4mlir-translate
P4MLIR dialect
Backend
BMv2
Tofino
LLVM dialect?
CIRCT dialects?
P4HIR
P4 High-level IR
New SW/HW targets?
LLVM IR
Verilog
Integration with external projects
External MLIR dialect
LLVM COMPILER INFRASTRUCTURE
MLIR
CIRCT

P4MLIR Today

- Makes control/data flow explicit while preserving high-level P4 semantics
- Basic statements and translations from P4 frontend IR are already implemented
- Starting to implement existing P4 frontend/midend passes

```
void doubleIfEven(inout bit<8> val) {
      bool isEven = val % 2 == 0;
      if (isEven) {
            val = val * 2;
      }
}
```

P4MLIR Today

- Makes control/data flow explicit while preserving high-level P4 semantics
- Basic statements and translations from P4 frontend IR are already implemented
- Starting to implement existing P4 frontend/midend passes

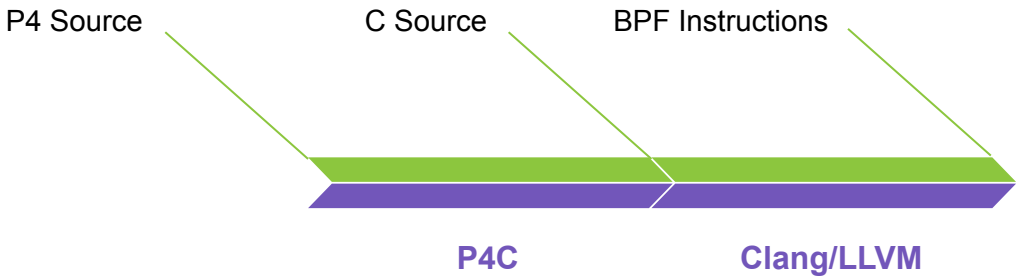
```
p4hir.func @doubleIfEven(%arg0: !p4hir.ref<!b8i>
{p4hir.dir = #p4hir<dir inout>, p4hir.param_name =
"val"}) {
      %c2_b8i = p4hir.const #int2_b8i
      %val = p4hir.read %arg0
      %mod = p4hir.binop(mod, %val, %c2_b8i)
      %eq = p4hir.cmp(eq, %mod, %c0)
      %isEven = p4hir.variable ["isEven", init]
      p4hir.assign %eq, %isEven
      %val_0 = p4hir.read %isEven
      p4hir.if %val_0 {
            %c2_b8i_1 = p4hir.const #int2_b8i
            %val_2 = p4hir.read %arg0
            %mul = p4hir.binop(mul, %val_2, %c2_b8i_1)
            p4hir.assign %mul, %arg0
      }
      p4hir.return
  }
```

Midpoint

- Context
  - There are 57 frontend and midend passes in the entire P4C repo
  - Some frontend passes are required to perform type inference
  - Other passes can be skipped as they are not needed
  - Every pass in P4C needs to copy the entire program, even on small changes
- Reach functional parity with existing P4C frontend and midend passes within the MLIR framework

Deliverables

- Address core performance issues
  - Improve data structure usage and existing implementations of passes
  - Design a new replacement use-def analysis pass with MLIR
- Lay groundwork for backend extensibility by lowering toward LLVM IR



Questions

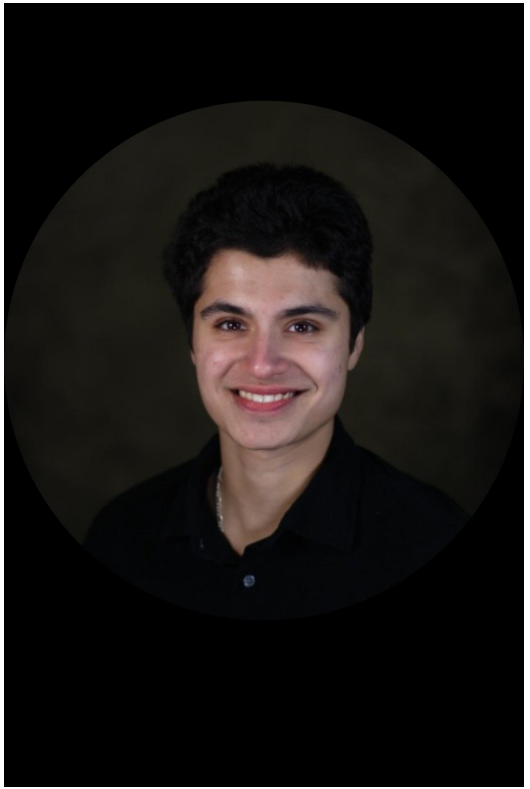
Gigaflow: A Smart Cache for a SmartNIC

Advay Singh

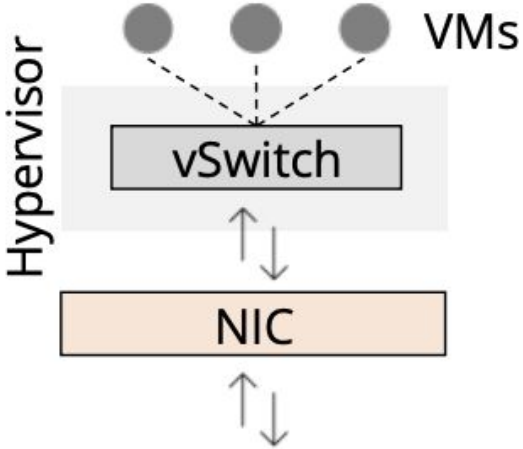
Advay Singh

Pursuing my Bachelors in Computer Science at the University of Michigan, USA

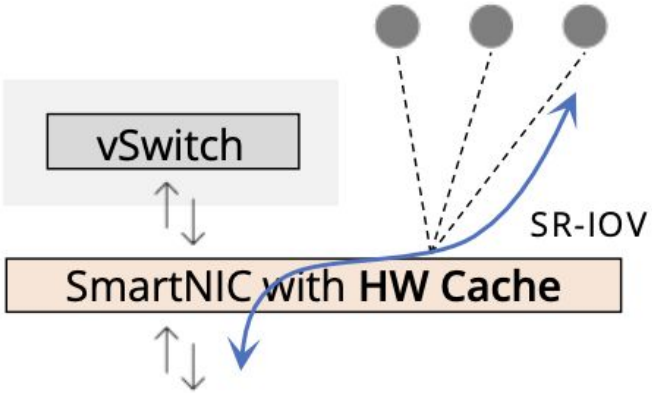
- First time open source contributor!
- Interested in systems, architecture, networks, and compilers
- Github: https://github.com/AdvaySingh1
- Hobbies: painting, boxing



Open vSwitch And HW Offload



Slow and CPU intensive software flow lookup…



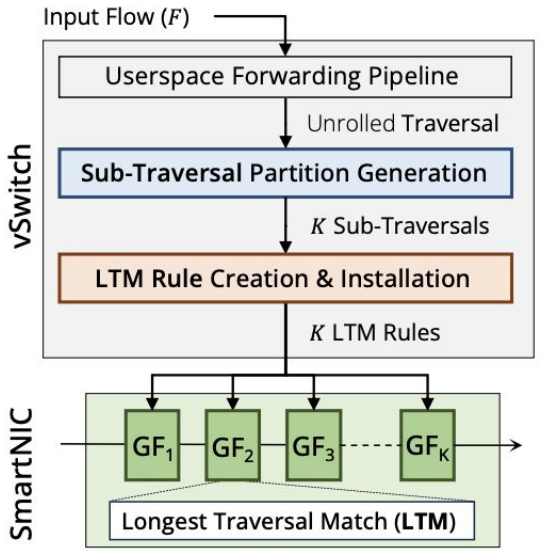
Limitations in space makes it difficult to cache many flows, resulting in low HW cache hits

Gigaflow

Leveraging
1) Pipeline locality
2) Rule space reusability,

The Gigaflow cache archives higher cache hit rates

Key Idea: Break Openflow pipeline traversals into multiple sub-traversals which flows can share



Gigaflow Architecture

Gigaflow & P4

A Gigaflow cache is a pipeline of p4 tables full of Longest Traversal Matching (LTM) rules

```
table G0 {
    // every table MUST EXACT MATCH on, at least, the table tag
    key            = { meta.table_tag    : exact;   // table_tag
                       meta.src_port     : ternary;   // in_port
                       hdr.eth.smac      : ternary;   // dl_src
                       hdr.eth.dmac      : ternary;   // dl_dst
                       hdr.eth.type      : ternary;   // dl_type
                       hdr.ipv4.src      : ternary;   // nw_src
                       hdr.ipv4.dst      : ternary;   // nw_dst
                       hdr.ipv4.protocol : ternary;   // nw_proto
                       meta.tp_src       : ternary;   // tp_src
                       meta.tp_dst       : ternary;   // tp_dst
                     }
    actions        = { forward;
                       drop;
                       insert_next_table_tag;
                       insert_next_table_tag_and_forward;
                       NoAction;
                       // ...
                       }
    size           = NUM_ENTRIES;
    num_masks      = 64;
    default_action = NoAction;
}
```

```
apply {
    G0.apply();
    G1.apply();
    G2.apply();
    G3.apply();
    if(meta.dst_port == 0){
        send_to_cpu();
    }
}
```

Challenges And Goal

Gigaflow is currently a SW emulation (no HW offload supported)

Implement the offload in an AMD Alveo U250 board

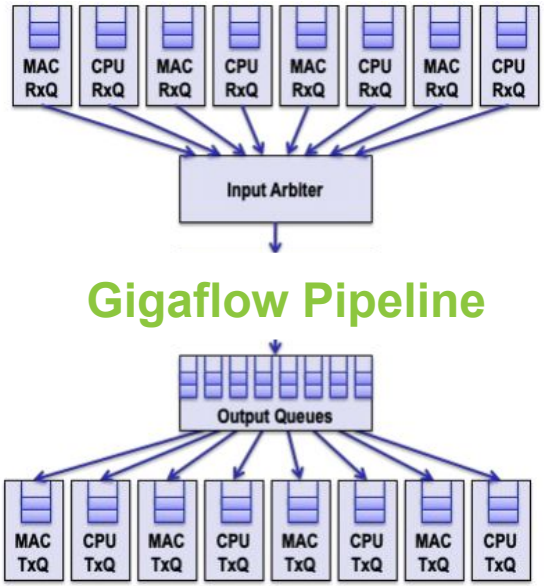
Main deliverable: Create a **template** for HW offload of Gigaflow caches on P4 enabled devices

AMD Alveo U250

Deliverables

Gigaflow Tables

AMD P4-SDNet



Gigaflow Pipeline

NetFPGA Shell

AMD Vivado

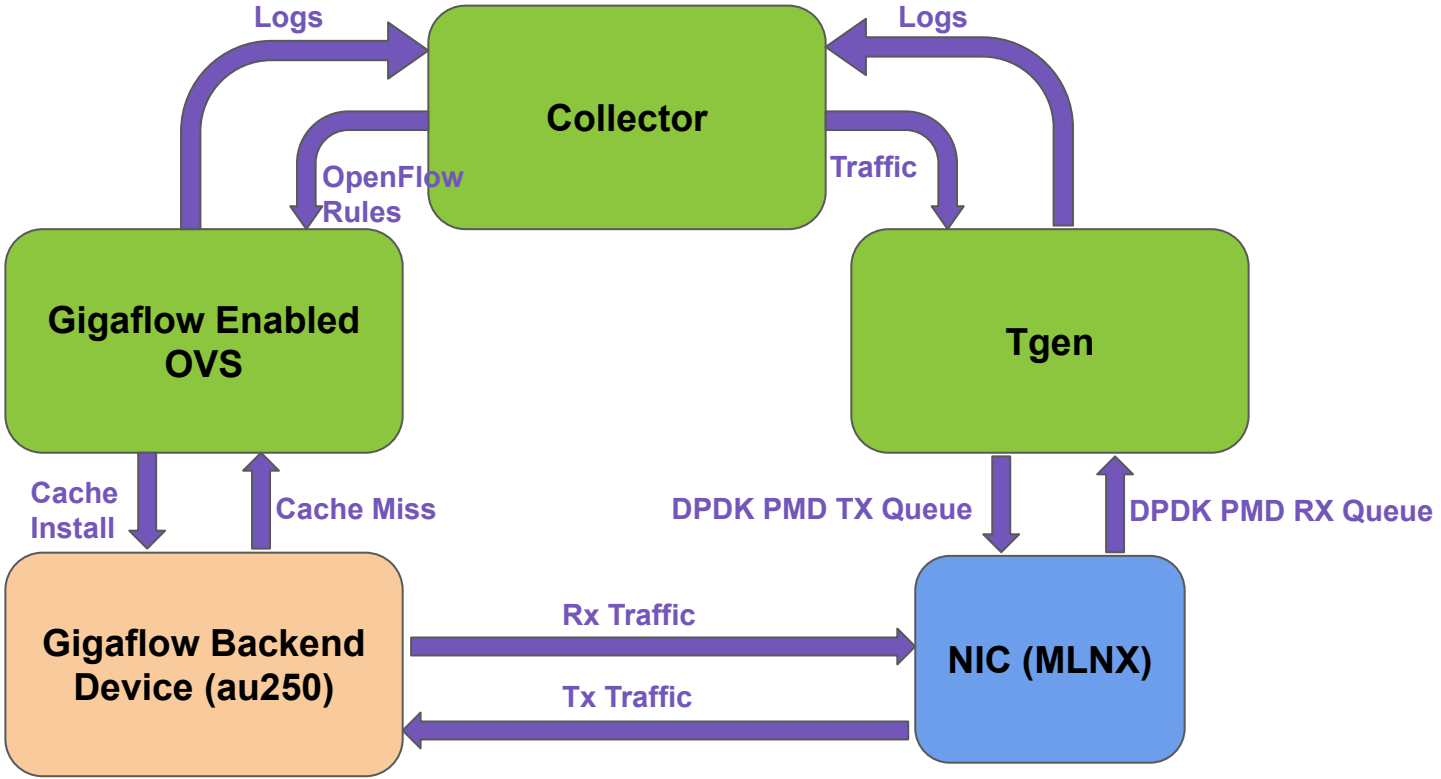
AMD Alveo U250

Deliverables Cont.

AMD SDNet Driver + 

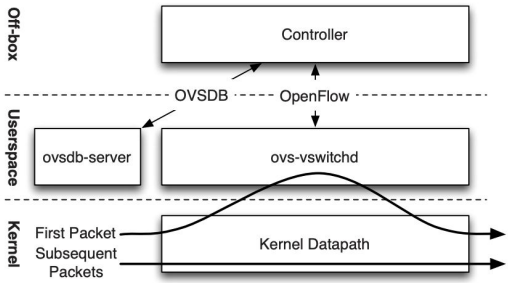
Install p4 rules into Gigaflow tables

Current Infrastructure



Value

- Open vSwitch is the most popular virtual switch in the world
- Its improvements have been incremental



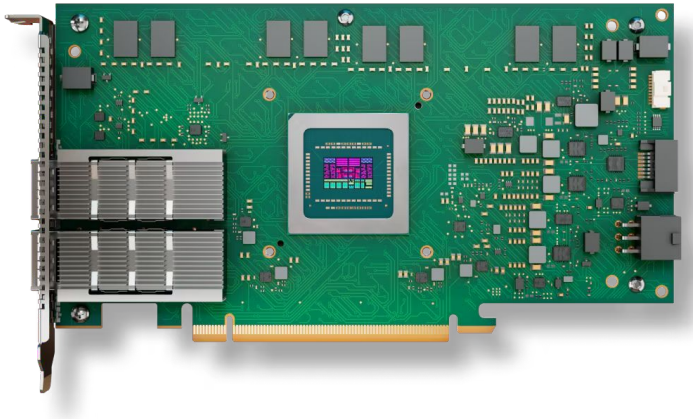
- Microflow Kernel Caches
- Megaflow Kernel Caches

- DPDK
- AF_XDP
- SR_IOV

- Now it’s time for Gigaflow

Future Work

Using the template, to add more p4 enabled devices (AMD Pensando, BlueField DPUs, etc.) to the Gigaflow backend



AMD Pensando

Nvidia BlueField

SpliDT: Scaling Stateful Decision Tree Algorithms in P4

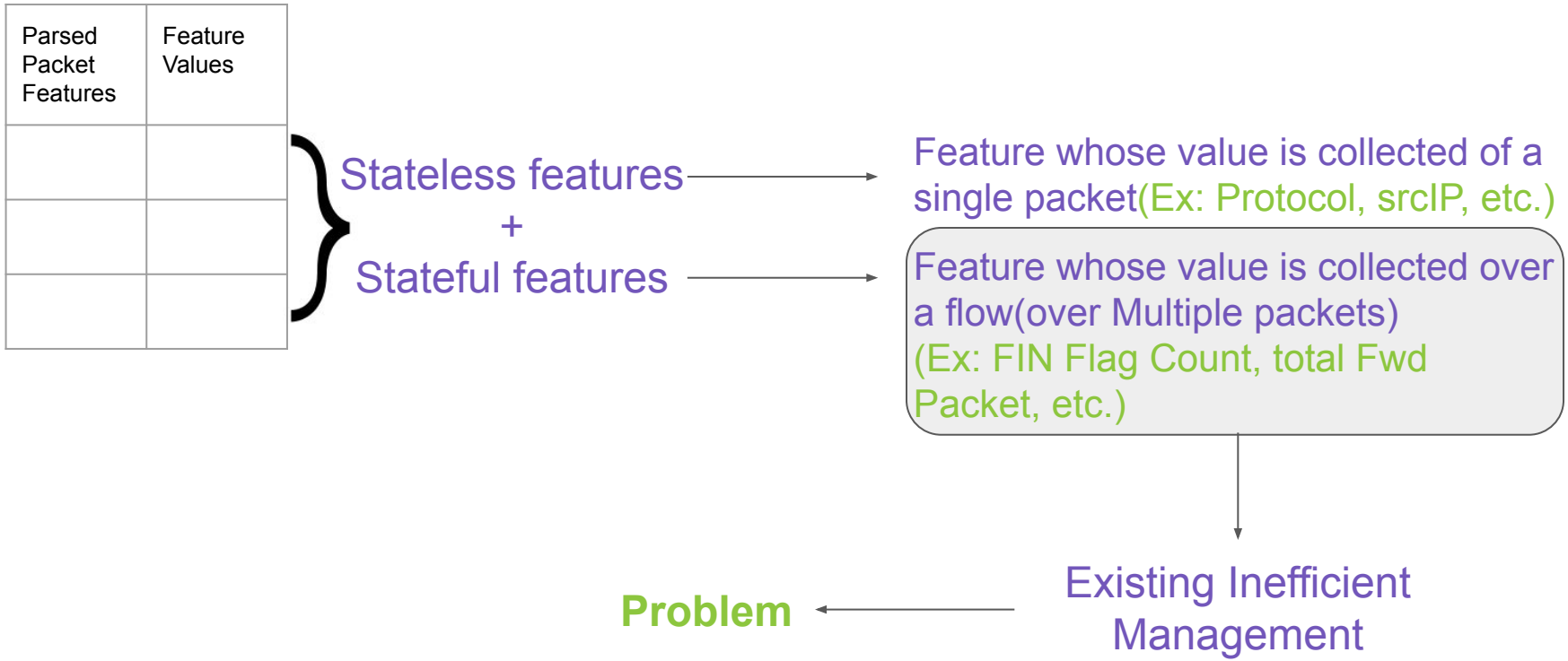
Sankalp Jha

Sankalp Jha

Bachelor of Technology in Computer Science & Information Technology,
Ajay Kumar Garg Engineering College, India

- Co-Organizer of Google Developer Groups on Campus AKGEC
- College Captain of Men’s Table Tennis Team
- Public Speaker & Content Writer
- Debuting as an Open Source Contributor

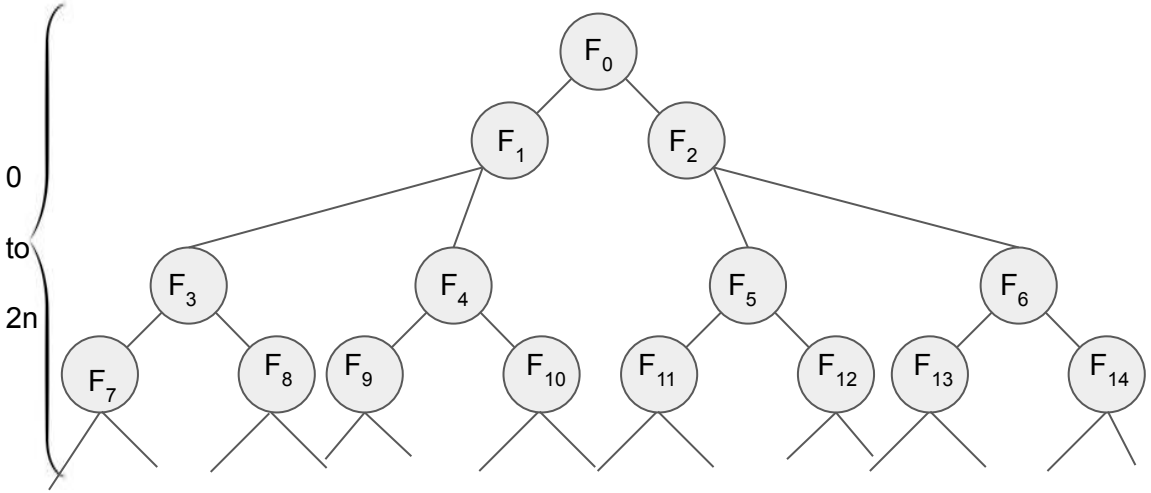
Problem Statement



Problem Statement Contd.

A flow having 2n packets

Conventional Decision Tree



No. of Features(n)

Feature 1 Flow 1	Feature 2 Flow 1	Feature 3 Flow 1
Feature 1 Flow 2	Feature 2 Flow 2	Feature 3 Flow 2
Feature 1 Flow 3	Feature 2 Flow 3	Feature 3 Flow 3
Feature 1 Flow 3	Feature 2 Flow 3	Feature n Flow m

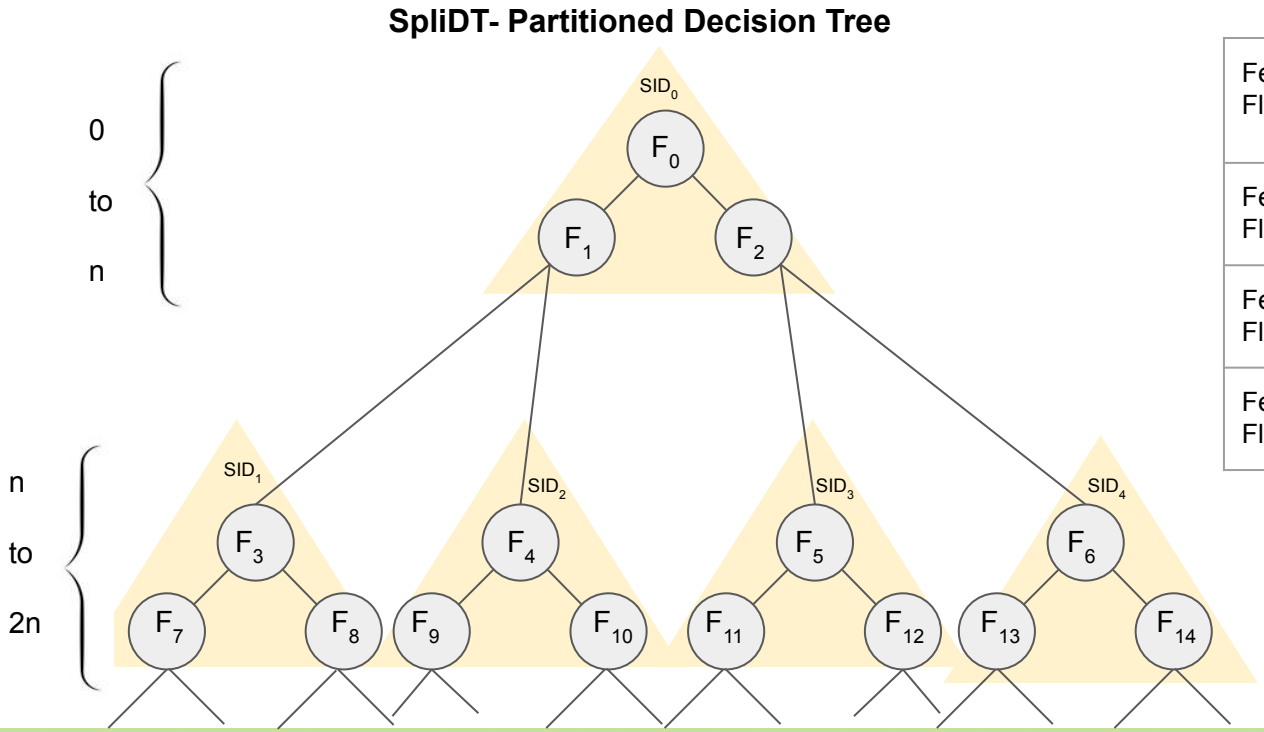
No. of Flows (m)

Stateful Memory

No. of Features Increases -> No. of column Increases
No. of Flow Decreases -> No. of Row Decreases
Vice-versa too ->Area is constant-> n * m = constant

Proposed Solution

A flow having 2n packets



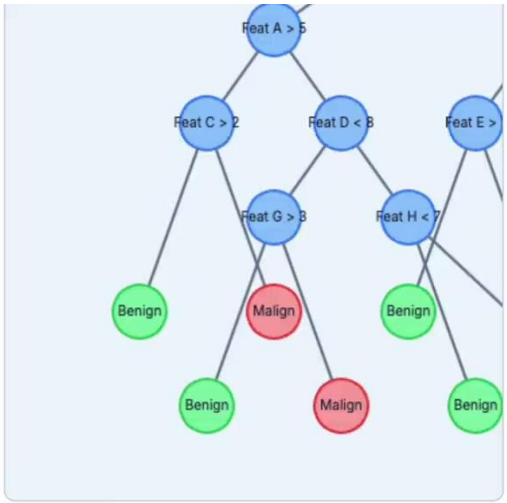
No. of Features(n)

Feature 1 Flow 1	Feature 2 Flow 1	Feature 3 Flow 1
Feature 1 Flow 2	Feature 2 Flow 2	Feature 3 Flow 2
Feature 1 Flow 3	Feature 2 Flow 3	Feature 3 Flow 3
Feature 1 Flow 3	Feature 2 Flow 3	Feature n Flow m

No. of F (m)

Stateful Memory

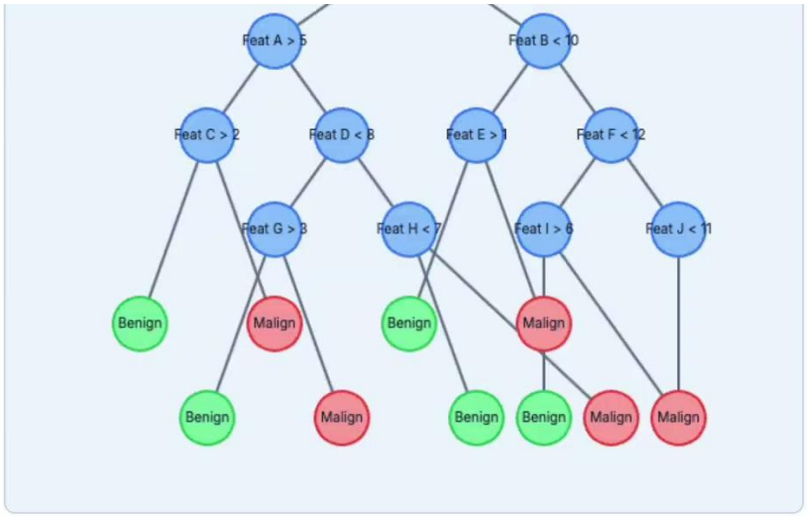
Proposed Solution Contd.



Current Features

Feature	Value
No features active	

Start Pause Reset

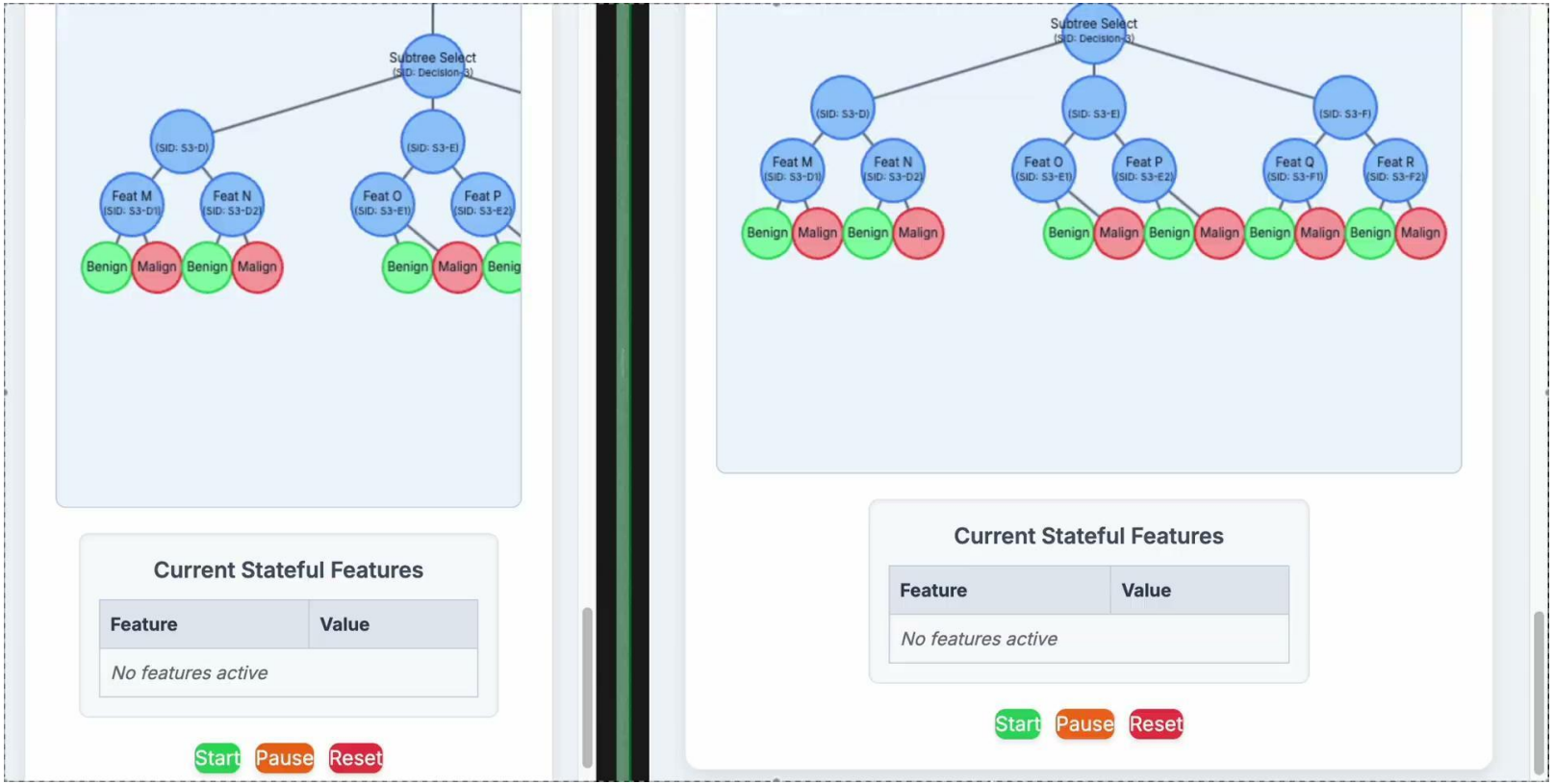


Current Features

Feature	Value
No features active	

Start Pause Reset

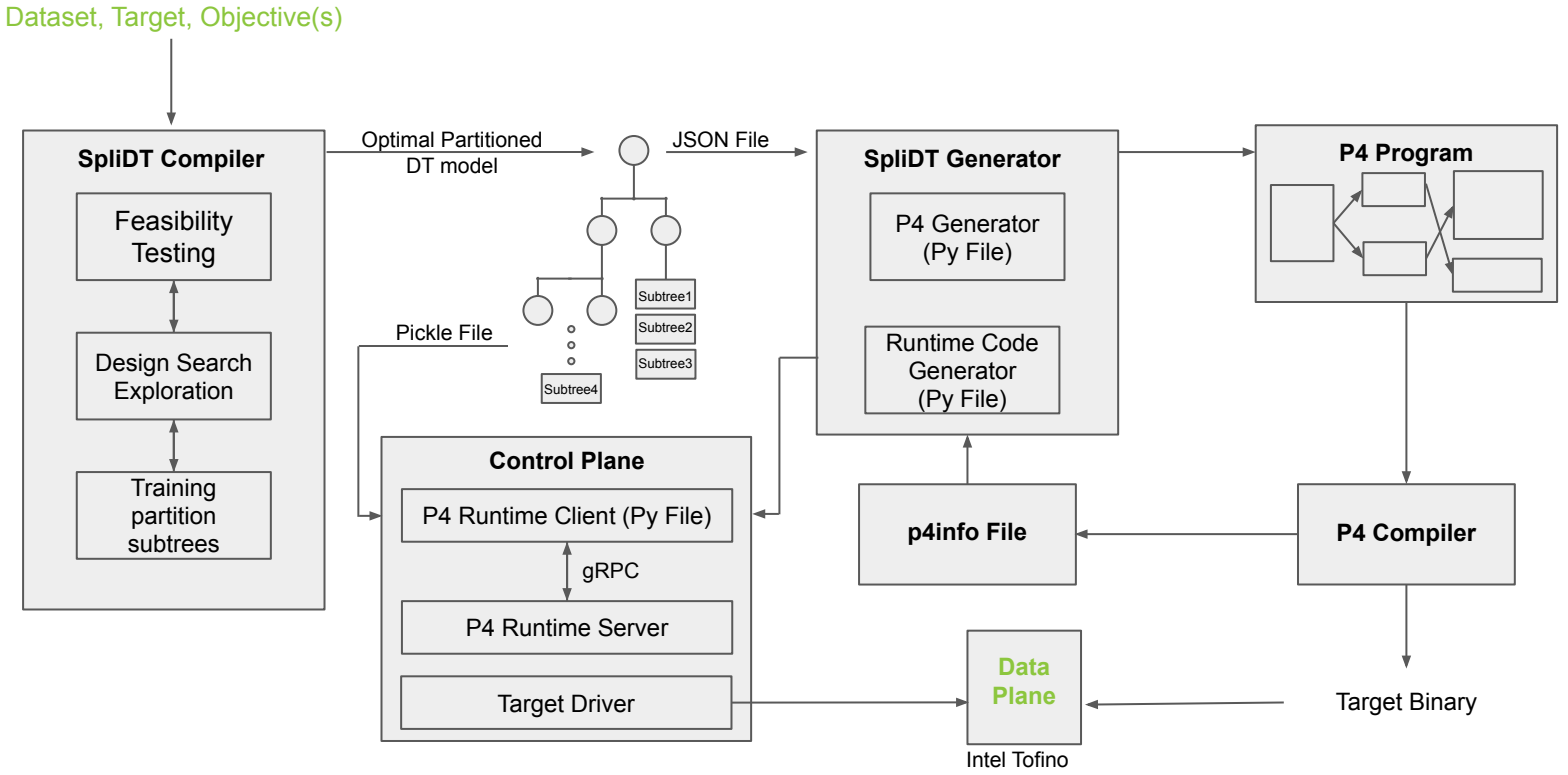
Proposed Solution Contd.



Challenges & Goals

- Scale decision tree inference under strict switch resource limits (TCAM, SRAM, PHV)
- Support dynamic, partitioned feature selection instead of fixed top-K features
- Convert trained models into efficient, deployable P4 match-action logic
- Automate the full flow from dataset training to switch deployment and testing

Project Pipeline



Deliverables

A. Software-based Action Items
(Completed)

B. P4 Data-Plane Action Items
(In Progress)

Re-ran decision tree training with HyperMapper config tuning.
Integrated CICFlowMeter for consistent feature extraction.

Extracted top-performing partition DT with SID-feature mappings.
Generated TCAM/SRAM entries for controller deployment.

Built SpliDT-based inference prototype with fixed features.
Validated memory layout, register access patterns.

Running simulations on Tofino with synthetic traffic.
Planning auto-generation of subtree-specific P4 code.

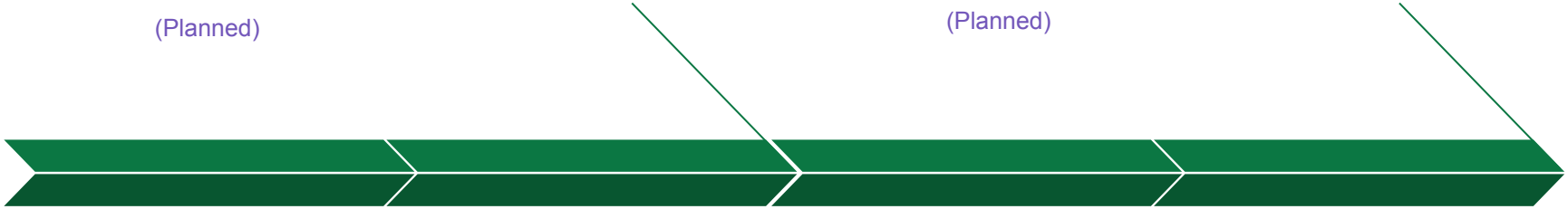
Deliverables

C. Controller Action Items

(Planned)

D. Automation & Deployment

(Planned)



Define P4Runtime protocols and gRPC-based control flow.
Handle flow-to-subtree dynamic mapping

Consume codegen output (p4info/context) for programming.
Deploy decision tree partitions into switch MATs.

Create end-to-end pipeline from dataset to live switch.
Automate training, codegen, and deployment steps.

Use MoonGen for traffic aligned to dataset behavior.
Deploy via Ansible and collect performance metrics.

Future Scope

- Generalize the pipeline to support other programmable platforms like SmartNICs or FPGA-based switches.
- Dynamic subtree reconfiguration via control-plane triggers.
- Production-Scale Integration in data centers for tasks like DDoS detection, QoS enforcement, or flow classification.
- Extend beyond Decision Trees to support other interpretable models like Random Forests in P4.

Q&A Session

Grateful for this opportunity to excel