Composing Dataplane Programs with \( \mu P_4 \)

Hardik Soni
Myriana Rifai, Praveen Kumar, Ryan Doenges, Nate Foster
Modular Programming

Networks increasingly look like any other software system...

- P₄ programs
- Control plane programs
- SDN
- Dataplane

Compiler
“Modularity based on abstraction is the way things are done”

2008 Turing Award Speech

Barbara Liskov
Why is modularity important?

1. Decompose large systems into small pieces that can be developed independently

2. Develop libraries of common code fragments that can be reused in many different systems

3. Write high-level code once, let a compiler port to different target devices
How are we building network software today?
How are we building network software today?

switch.p4

The switch.p4 program describes a data plane of an L2/L3 switch.

Supported Features

1. Basic L2 Switching: Flooding, learning and STP
2. L2 Multicast
3. Basic L3 Routing (unicast); IPv4 and IPv6 and VRF
4. L3 Multicast
5. LAG
6. ECMP
7. Tunneling: VXLAN and NVGRE (including L2/L3 Gateway), Geneve, GRE and IPinIP
8. Basic ACL: MAC and IP ACLs
9. Unicast RPF check
10. MPLS: LER, LSR, IPVVPN, VPLS, L2VPN
11. Host interface
12. Mirroring: Ingress and egress mirroring with ERSPAN
13. Counters/Statistics
14. Ingress Policers
15. Inband Network Tlemetry (INT)

Upcoming Features

1. NAT
2. QoS
How are we building network software today?

```
122  control process_ipv6_fib {
123      #if !defined(L3_DISABLE) && !defined(IPV6_DISABLE)
124          /* fib lookup */
125          apply(ipv6_fib) {
126            on_miss {
127                apply(ipv6_fib_lpm);
128            }
129          }
130      #endif /* L3_DISABLE && IPV6_DISABLE */
131  }
```
```c
122  control process_ipv6_fib {
123    #if !defined(L3_DISABLE) && !defined(IPV6_DISABLE)
124       /* fib lookup */
125       apply(ipv6_fib) {
126          on_miss {
127              apply(ipv6_fib_lpm);
128          }
129       }
130    #endif /* L3_DISABLE && IPV6_DISABLE */
131  }
```
Wouldn’t it be nice if...

```c
{ 
    if (!defined(IPV6)) 
    { 
        on_miss()
        apply(ipv6_fib_lpm);
    }
}
```
Challenges
P₄ Programs are...

1. Monolithic

2. Written for a Heterogeneous Programming Model

3. Tightly Coupled to Target Architectures
1. P4 Programs are Monolithic

IPv4

IPv6

MPLS

Let’s see how we implement routing with P4 today ...
1. P4 Programs are Monolithic

Code snippet from switch.p4
2. Heterogeneous Programming Model

State Machine based Packet-parsing

Match-Action based Packet-processing

Fixed-function Externs
3. Tight-Coupling with Architectures

**header** Ethernet {bit<48> dstmac; ...}  
**struct** my_hdr_t {Ethernet eth; ...}

**parser** IngressParser(
  packet_in pin,
  out my_hdr_t h,
  inout my_meta_t m,
  ...) {
  // parser code goes here
}

**control** Ingress(
  inout my_hdr_t h,
  inout my_meta_t m,
  ...) {
  // match-action tables
}

**control** IngressDeparser(
  packet_out po,
  ...,  
  inout my_hdr_t h,
  ...,  
  inout my_meta_t m) {
  // packet reassembly
}

Portable Switch Architecture

Programmable Blocks  
Fixed-function blocks
3. Tight-Coupling with Architectures

Makes it difficult to port programs

```c
header Ethernet {bit<48> dstmac; ...}  
struct my_hdr_t {Ethernet eth; ...}  

parser IngressParser(  
  packet_in pin,  
  out my_hdr_t h,  
  inout my_meta_t m,  
  ...) {  
  // parser code goes here  
}

control Ingress(  
  inout my_hdr_t h,  
  inout my_meta_t m,  
  ...) {  
  // match-action tables  
}

control IngressDeparser(  
  packet_out po,  
  ...,  
  inout my_hdr_t h,  
  in my_meta_t m) {  
  // packet reassembly  
}
```

Spectrum Mellanox:

- Parser
- Port
- Flex1
- Bridge
- Flex2
- Router
- Flex3
- Tunnel
- Deparser

Programmable Blocks

Fixed-function blocks
μP4
Design Insights

1. Higher-Level Abstractions for Dataplanes

2. Homogenize the Programming Model
μP4: Abstract Dataplane Model

Key Ingredients:

● **Logical pipeline:** distill packet processing down to its essence, as a three-stage read-process-write function

● **Logical buffers:** Provide a common interface for composing
What μP₄ Abstracts?

<table>
<thead>
<tr>
<th>P₄</th>
<th>μP₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture pipeline</td>
<td>Logical micro-pipelines</td>
</tr>
<tr>
<td>Architecture metadata for packet processing</td>
<td>Logical Buffer</td>
</tr>
<tr>
<td>Architecture-specific fixed-functions</td>
<td>Logical Externs</td>
</tr>
</tbody>
</table>

Diagram:
- Ingress pipeline:
P4: Parser → Control → Deparser → Packet Buffer → Parser → Control → Deparser
- Egress pipeline:
P4: Parser → Control → Deparser
- μP₄ Module:
P4: Parser → Control → Deparser
μP4 supports Composition

Composed μP4 Module
μP₄ Module

Parser → Control → Deparser
Compiling μP4
Overview

- Source-to-source P4_16 compiler
- Based on open-source p4c framework
- Implemented as 13.5 KLoC of C++
- Backends for Bmv2 and Tofino

Technical Approach:

- Homogenize source program to facilitate analysis and transformation
- Rearrange code to respect target constraints
- Emit code for target's specialized packet-processing units
μP4 Compiler: Passes

μP4 logical pipeline

μp4 source

Frontend

Transformation to match-action control

Parser → Control → Deparser

Midend

Target-specific translation & control-block allocation

Parser → Control → Deparser

Backend

Target: Portable Switch Architecture

P4 source for target (e.g., PSA)

ingress

Packet Buffer

egress
Midend: Homogenize Abstract Machines

State machine based Packet-parsing

Match: Bytes in packet
Action:
1. Extract/Copy Bytes
2. Transition to next state

Match-Action based Packet-processing
Backend: Mapping to Target Architecture

Target: Portable Switch Architecture (PSA)
Evaluation
Questions

1. **Expressiveness**
   - Case study, developed library of common functions

2. **Portability**
   - Can run *same* programs on BMv2 and Tofino

3. **Efficiency**
   - Compared resource utilization against hand-written monolithic Tofino programs
# Expressiveness: μP4 Modules

<table>
<thead>
<tr>
<th>μP4 Modules</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL</td>
<td>120</td>
</tr>
<tr>
<td>ETH</td>
<td>70</td>
</tr>
<tr>
<td>IPv4</td>
<td>88</td>
</tr>
<tr>
<td>IPv6</td>
<td>73</td>
</tr>
<tr>
<td>MPLS</td>
<td>124</td>
</tr>
<tr>
<td>NAT</td>
<td>112</td>
</tr>
<tr>
<td>NPTv6</td>
<td>75</td>
</tr>
<tr>
<td>SRv4</td>
<td>173</td>
</tr>
<tr>
<td>SRv6</td>
<td>181</td>
</tr>
</tbody>
</table>
# Expressiveness: Composition

<table>
<thead>
<tr>
<th>μP4 Modules</th>
<th>Composed Programs</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ETH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPv4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPv6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MPLS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NAT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NPTv6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SRv4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SRv6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Example: Modular Router

<table>
<thead>
<tr>
<th>µP4 Modules</th>
<th>ACL</th>
<th>ETH</th>
<th>IPv4</th>
<th>IPv6</th>
<th>MPLS</th>
<th>NAT</th>
<th>NPTv6</th>
<th>SRv4</th>
<th>SRv6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Same program; multiple targets (BMv2 and Barefoot’s Tofino)
## Resource Overhead with Barefoot Tofino

<table>
<thead>
<tr>
<th>µP4 Modules</th>
<th>Composed Programs</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ETH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPv4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPv6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MPLS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NAT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NPTv6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SRv4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SRv6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#MAUs</th>
<th>P4 Monolithic</th>
<th>3</th>
<th>4</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>µP4 Composed</td>
<td>5</td>
<td>9</td>
<td>8</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
Wrapping Up...
Ongoing Work

Working to extend our current µP4 prototype to support:

- Packet replication and multicast
- Variable length headers
- Stateful packet-processing
- Control-plane APIs

https://github.com/cornell-netlab/MicroP4
Takeaways

Liskov was right: “Modularity based on abstraction is how things are done”

- **Language Design**
  - High-level abstractions for data plane programming

- **Compiler**
  - Composes different modules into single program
  - Generates code for underlying targets

- **Experience**
  - Developed a modular router

μP4 enables portable, modular, & composable data plane programming!