POLYTECHNIQUE
MONTRÉAL

TECHNOLOGICAL
UNIVERSITY

P4Muse: Enabling Modular P4 Programming via Compiler-Managed Code Merging Without Syntax Modifications

Mohsen Rahmati and François-Raymond Boyer

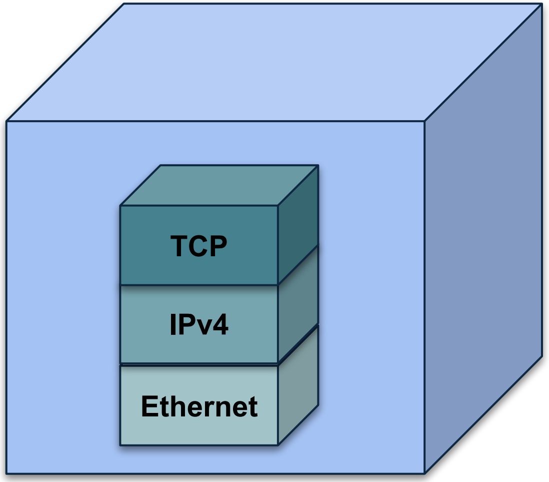
Authors: Mohsen Rahmati, François-Raymond Boyer, Bill Pontikakis, Yvon Savaria, Jean Pierre David

March 04, 2026

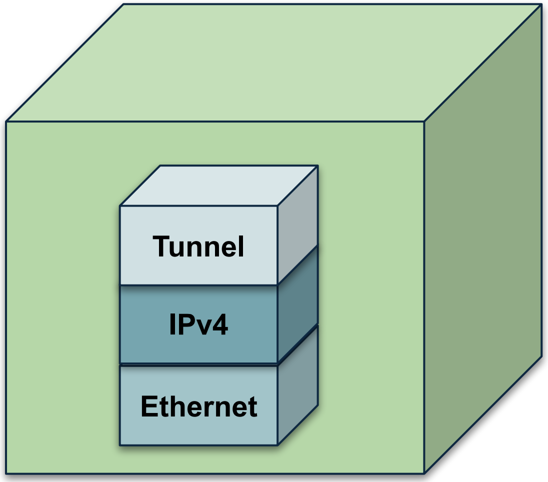
Outline

Problem Statement

- Current P4 programs are monolithic and hard to reuse



a) Firewall code



b) Advanced tunnel code

Figure 1. Firewall and advanced tunnel codes.

Problem Statement:
The Importance of Modularity in P4

- Modern data planes are large, complex, and rapidly evolving
    - Reduce programming time
    - Create less error-prone codes
- Modularity in protocol stack
- The benefit of using vendor code along with customer codes
    - Incremental programming

Problem Statement: Simplicity and Modularity in P4

Figure 6. The simplicity of P4 language [2].

[2] Bicycle, by dapple-designers. Pixabay. https://pixabay.com/vectors/cycle-bicycle-load-green-bike-4536914/ (Published Oct 10 2019). Used under the Pixabay Content License.

Related Works

µP4 [3]	Homogenizes everything into match–action,impacting architecture, IR, and syntax
Lyra [4]	Hides low-level P4 code with a One-Big-Pipeline, impacting architecture, IR, and syntax
daPIPE [5]	provides a GUI for incremental P4 programming
P4Weaver [6]	Use annotation system and ANTLR as parser generator for incremental programming
P4Ansible [7]	Non-complete work, add new syntax for inheritance but not in IR

[3] Soni — µP4, SIGCOMM ’20 [4] Gao — Lyra, SIGCOMM ’20 [5] Baldi — daPIPE, ANCS ’19 [6] Fattaholmanan — P4 Weaver, SOSR ’21
[7] MNK Lans — p4-ansible, Online (2021).

Related Works

	Automatic Merging	Backward Compatibility	Vendor-custom er Compatibility	No New Syntax or Annotation	Interaction with P4 IR	Inheritance
μP4 [3]	No	No	N/A	No	No	No
Lyra [4]	No	N/A	N/A	No	No	No
P4Weaver [6]	No	Yes	Yes	No	No	No
P4Ansible [7]	No	Yes	Yes	No	No	Yes
daPIPE [5]	No	Yes	Yes	No	No	No

Table 2. Feature Comparison of P4 Modularity Solutions

Research Questions

RQ1: Can modularity be introduced in the P4 language using its existing compiler and language without adding new syntax or annotation?

RQ2: Can object-oriented style inheritance be added to P4 and give more modularity?

Research Objectives

- Support P4 modularity by merging P4 codes instead of modifying its syntax while offering backward compatibility
    - Parser modularity
    - Pipeline modularity

Modular Code Parser for the P4 Language

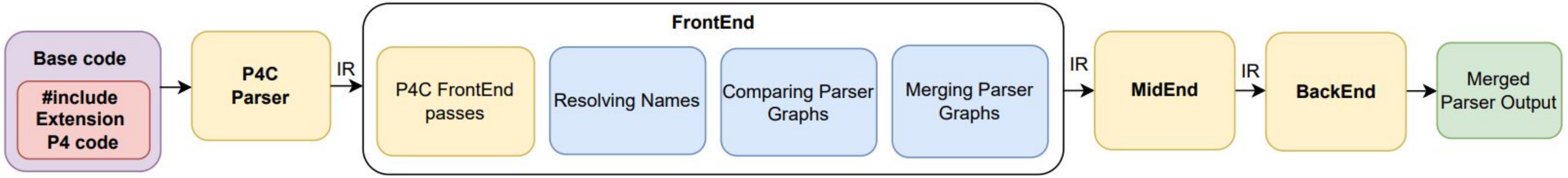
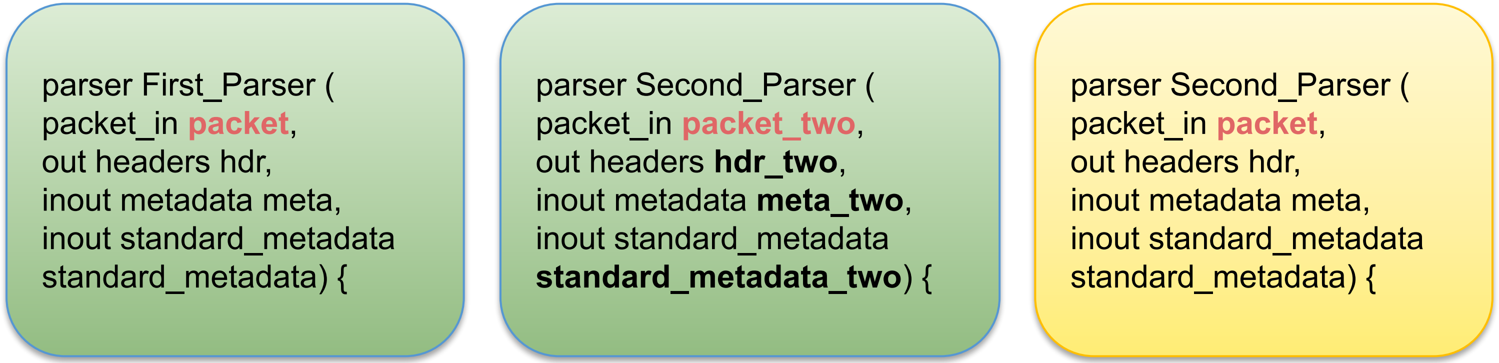


Figure 3. Overview of the modular parser compiler.

Resolving names



```
parser First_Parser (
packet_in packet,
out headers hdr,
inout metadata meta,
inout standard_metadata
standard_metadata) {
```

```
parser Second_Parser (
packet_in packet_two,
out headers hdr_two,
inout metadata meta_two,
inout standard_metadata
standard_metadata_two) {
```

```
parser Second_Parser (
packet_in packet,
out headers hdr,
inout metadata meta,
inout standard_metadata
standard_metadata) {
```

Figure 4. Resolving names for the second parser

Outcome

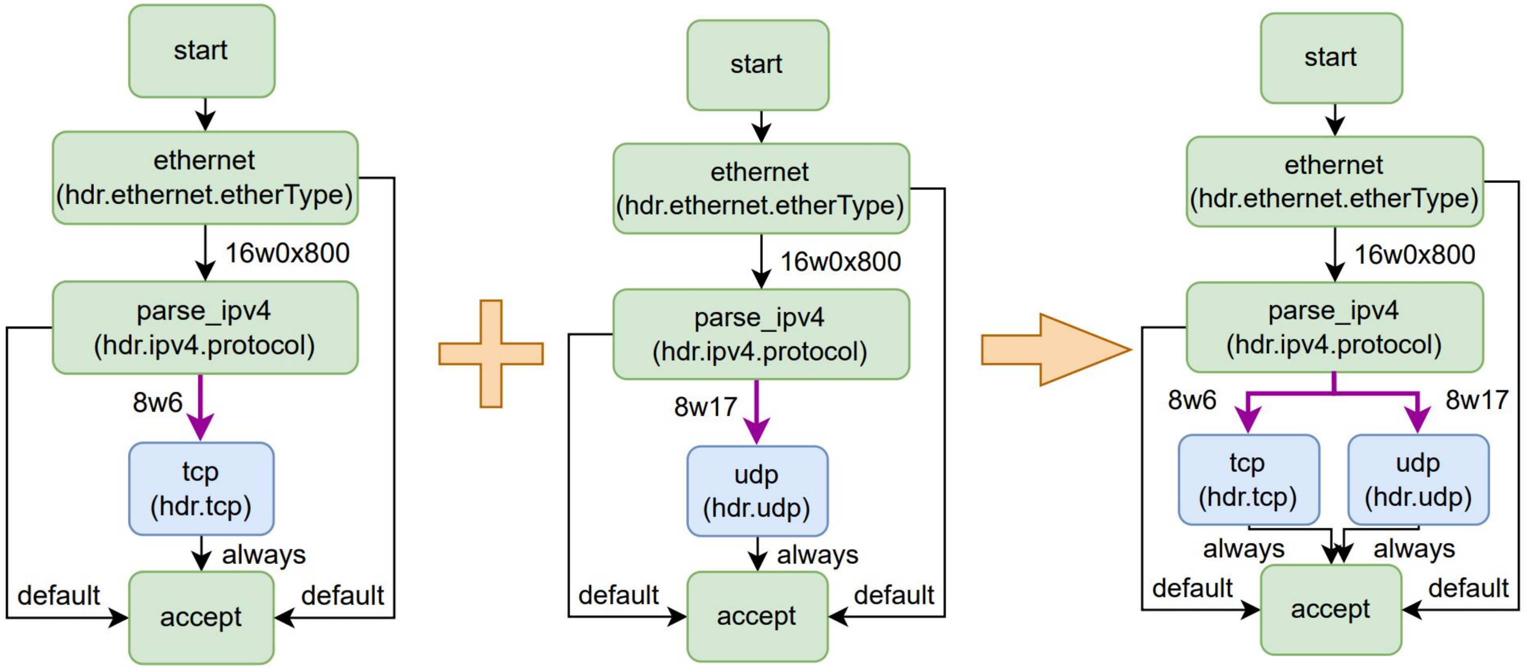

start
ethernet (hdr.ethernet.etherType)
16w0x800
parse_ipv4 (hdr.ipv4.protocol)
8w6
tcp (hdr.tcp)
always
default
accept
default
start
ethernet (hdr.ethernet.etherType)
16w0x800
parse_ipv4 (hdr.ipv4.protocol)
8w17
udp (hdr.udp)
always
default
accept
default
start
ethernet (hdr.ethernet.etherType)
16w0x800
parse_ipv4 (hdr.ipv4.protocol)
8w6
8w17
tcp (hdr.tcp)
udp (hdr.udp)
always
always
default
accept
default

Figure 5. The graphs of TCP, UDP, merged parser using [8, 9].

[8] "the open-source p4 compiler". https://github.com/p4lang/p4c.
[9] M. Rahmati, F.-R. Boyer, B. Pontikakis, J.-P. David, and Y. Savaria, "Modular Code Parser for the P4 Language," presented at the P4 Workshop, 2023.

Overview of Compiler-managed P4 modularity

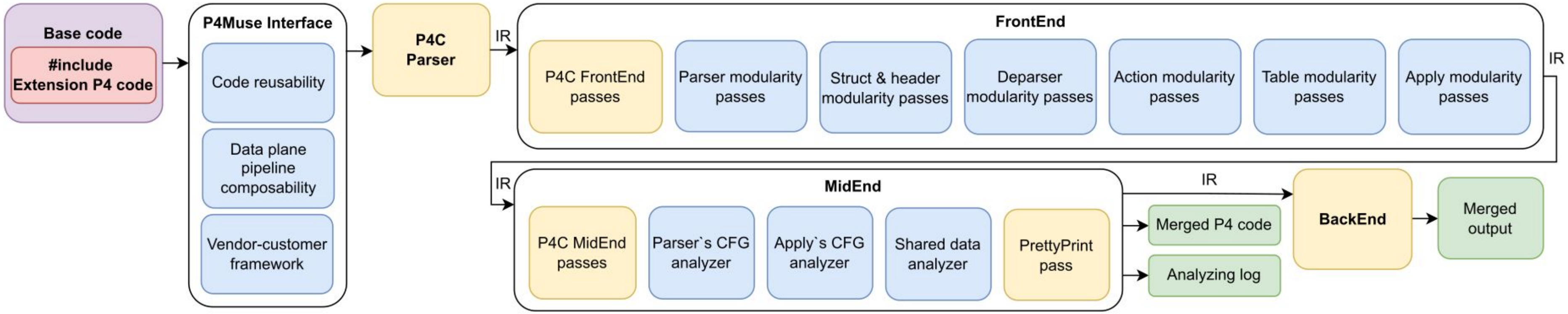


Figure 6. Overview of the P4Muse compiler showing its modular interface options and compiler stages [10].

[10] M. Rahmati, F. -R. Boyer, B. Pontikakis, J. Pierre David, and Y. Savaria, "P4Muse: Enabling Modular P4 Programming via Compiler-Managed Code Merging Without Syntax Modifications," in IEEE Access, vol. 13, pp. 124138-124157, 2025.

Parser Modularity

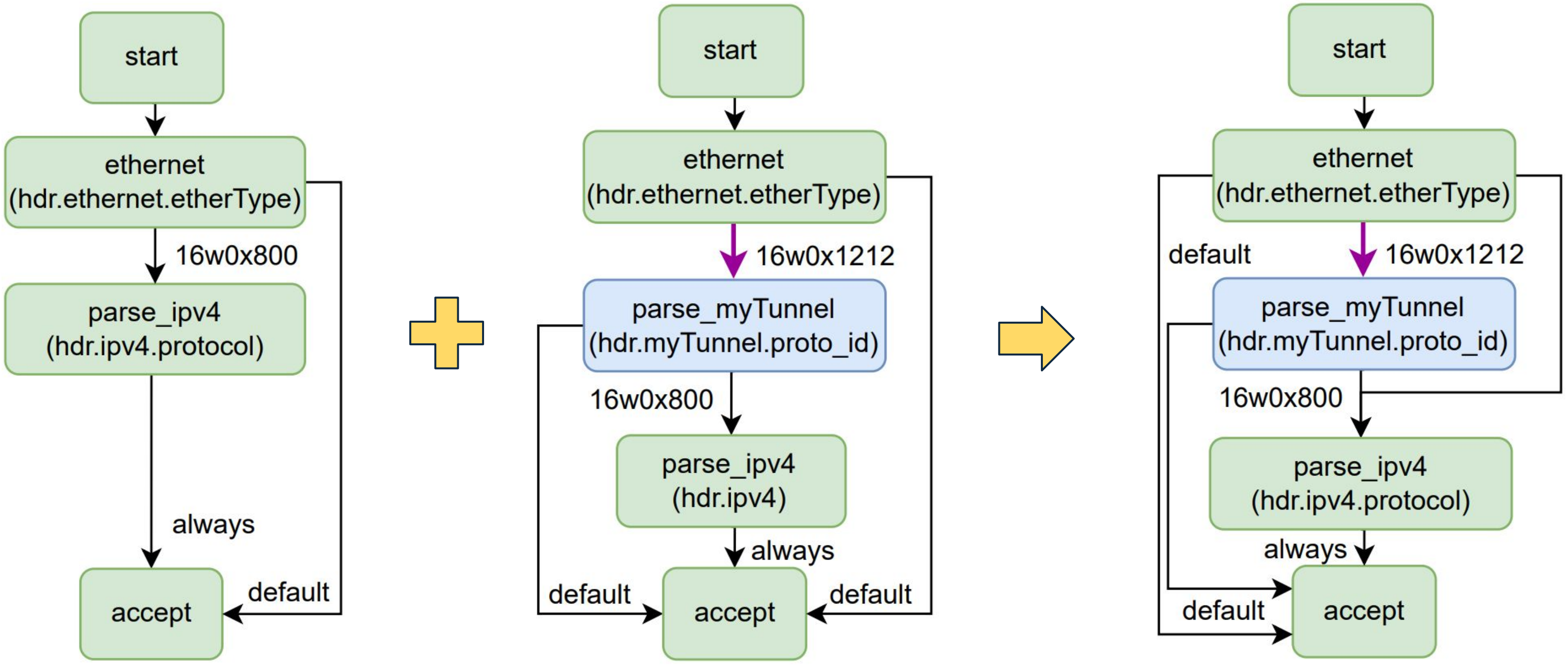
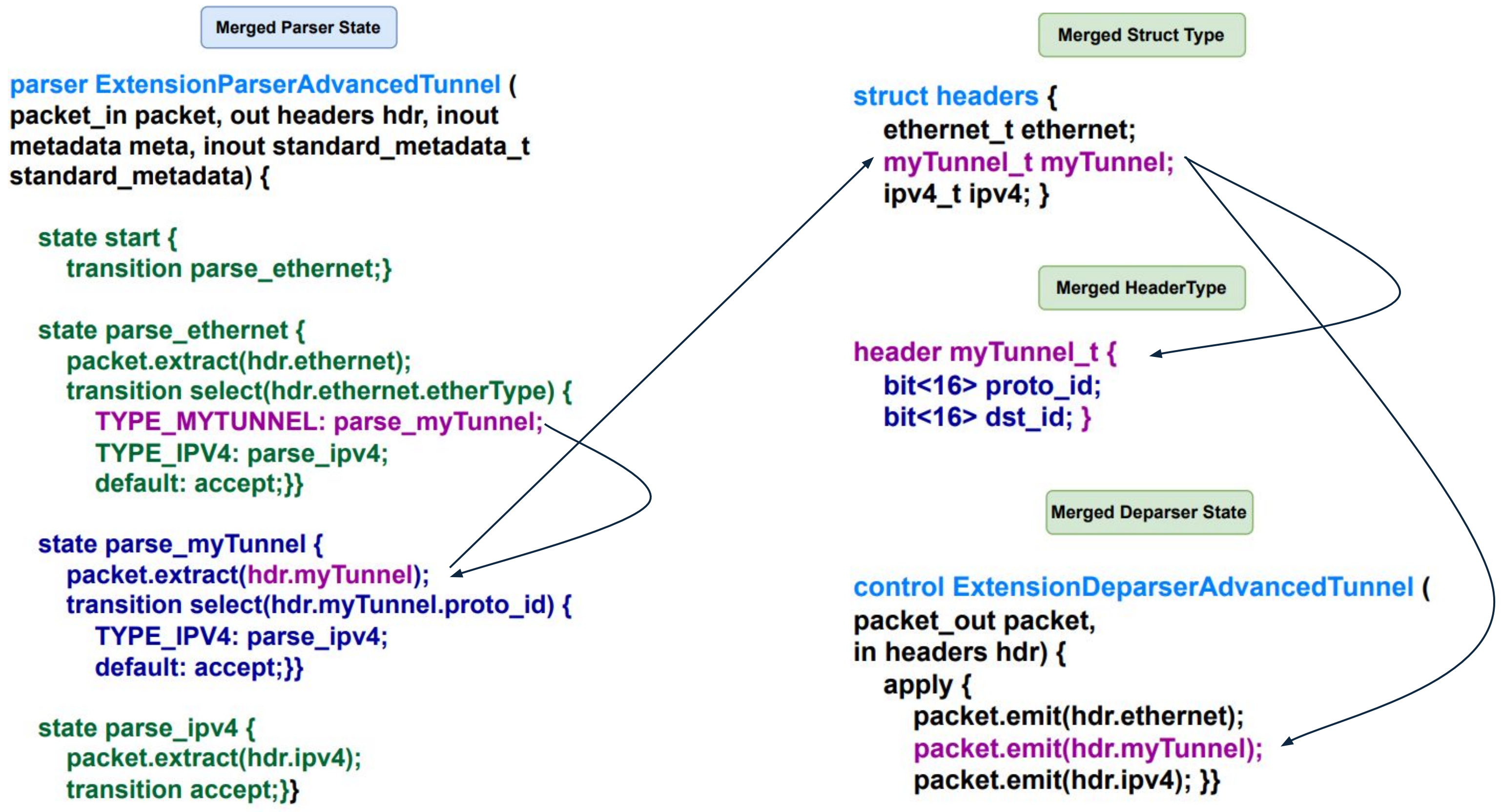

start
ethernet (hdr.ethernet.etherType)
16w0x800
parse_ipv4 (hdr.ipv4.protocol)
always
accept
default
start
ethernet (hdr.ethernet.etherType)
16w0x1212
parse_myTunnel (hdr.myTunnel.proto_id)
16w0x800
parse_ipv4 (hdr.ipv4)
always
default
accept
default
start
ethernet (hdr.ethernet.etherType)
default
16w0x1212
parse_myTunnel (hdr.myTunnel.proto_id)
16w0x800
parse_ipv4 (hdr.ipv4.protocol)
always
default
accept

Figure 7. Parser graphs of an IPv4, an advanced tunnel, and the corresponding merged P4 codes [10].

Struct, Header, and Deparser Modularity



```
parser ExtensionParserAdvancedTunnel (
packet_in packet, out headers hdr, inout
metadata meta, inout standard_metadata_t
standard_metadata) {

   state start {
      transition parse_ethernet;}

   state parse_ethernet {
      packet.extract(hdr.ethernet);
      transition select(hdr.ethernet.etherType) {
         TYPE_MYTUNNEL: parse_myTunnel;
         TYPE_IPV4: parse_ipv4;
         default: accept;}}

   state parse_myTunnel {
      packet.extract(hdr.myTunnel);
      transition select(hdr.myTunnel.proto_id) {
         TYPE_IPV4: parse_ipv4;
         default: accept;}}

   state parse_ipv4 {
      packet.extract(hdr.ipv4);
      transition accept;}}
```

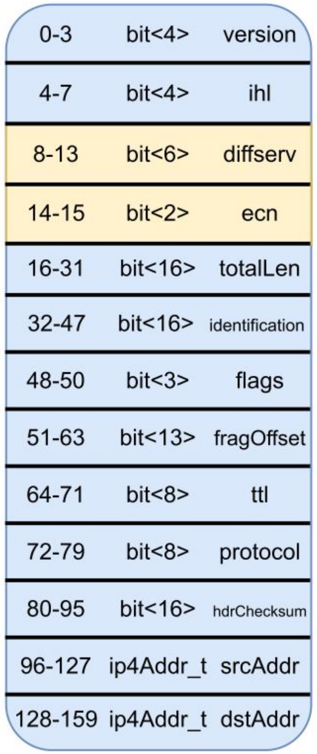
```
struct headers {
   ethernet_t ethernet;
   myTunnel_t myTunnel;
   ipv4_t ipv4; }
```

```
header myTunnel_t {
   bit<16> proto_id;
   bit<16> dst_id; }
```

```
control ExtensionDeparserAdvancedTunnel (
packet_out packet,
in headers hdr) {
   apply {
      packet.emit(hdr.ethernet);
      packet.emit(hdr.myTunnel);
      packet.emit(hdr.ipv4); }}
```

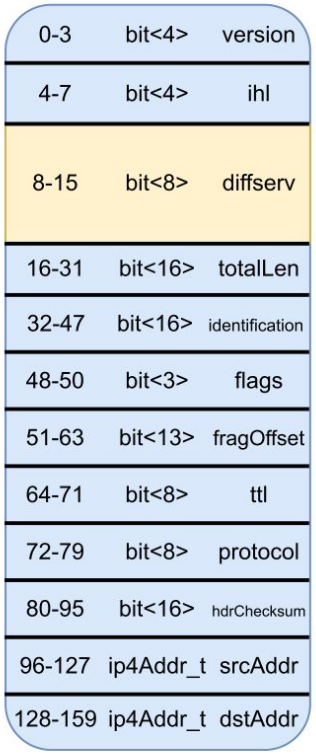
Figure 8. Parser, struct, header, and deparser blocks of advanced tunnel P4 code [10].

Header Modularity



0-3	bit<4>	version
4-7	bit<4>	ihl
8-13	bit<6>	diffserv
14-15	bit<2>	ecn
16-31	bit<16>	totalLen
32-47	bit<16>	identification
48-50	bit<3>	flags
51-63	bit<13>	fragOffset
64-71	bit<8>	ttl
72-79	bit<8>	protocol
80-95	bit<16>	hdrChecksum
96-127	ip4Addr_t	srcAddr
128-159	ip4Addr_t	dstAddr

IPv4 protocol with six bits diffserv



0-3	bit<4>	version
4-7	bit<4>	ihl
8-15	bit<8>	diffserv
16-31	bit<16>	totalLen
32-47	bit<16>	identification
48-50	bit<3>	flags
51-63	bit<13>	fragOffset
64-71	bit<8>	ttl
72-79	bit<8>	protocol
80-95	bit<16>	hdrChecksum
96-127	ip4Addr_t	srcAddr
128-159	ip4Addr_t	dstAddr

IPv4 protocol with eight bits diffserv

Figure 9. Comparison of IPv4 header formats with two kinds of DiffServ field lengths [10].

Table, Action, and Apply Modularity

- Action comparison & merging
- Table comparison & merging
- Apply-block comparison & merging

Modularity Level

- Construct-level Modularity
- Protocol-level modularity
- Application-level Modularity

Data Plane Pipeline Composability Framework

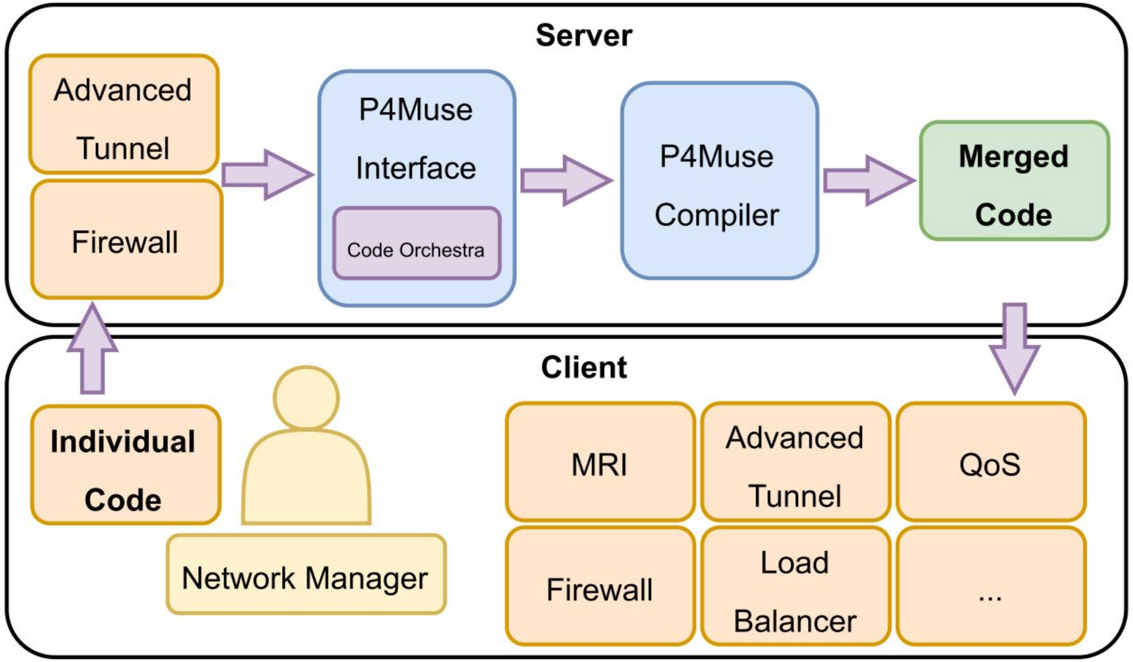


Figure 10. Client-server framework for data plane pipeline composability [10].

Vendor-customer Framework and Incremental Programming

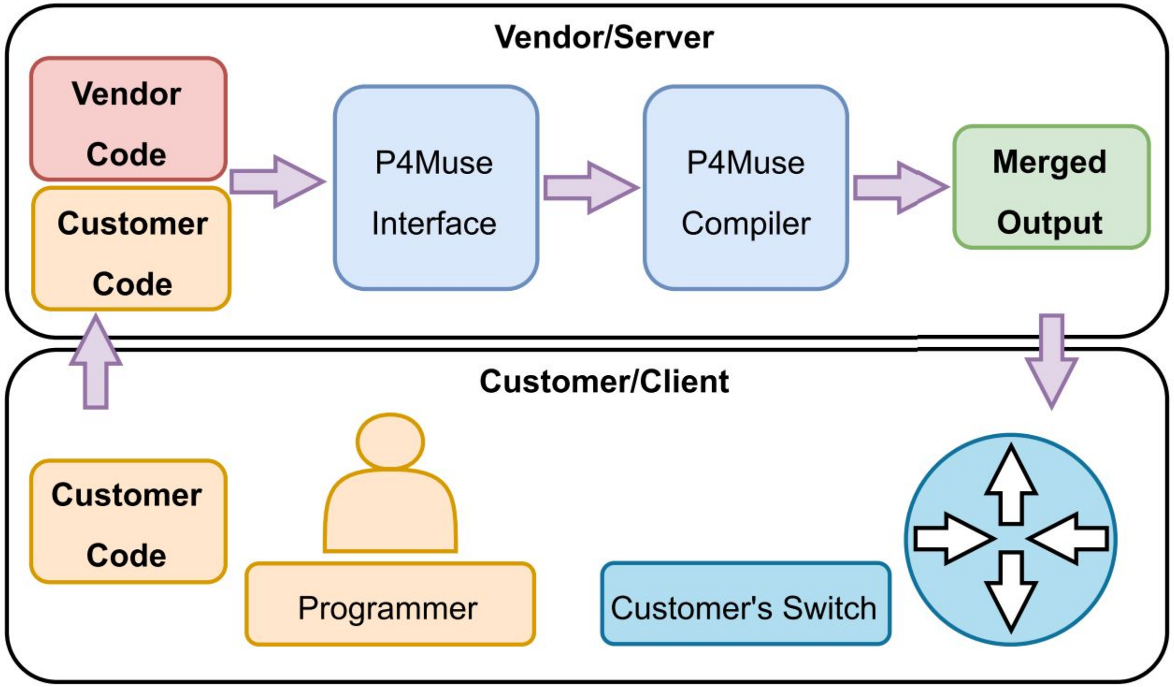
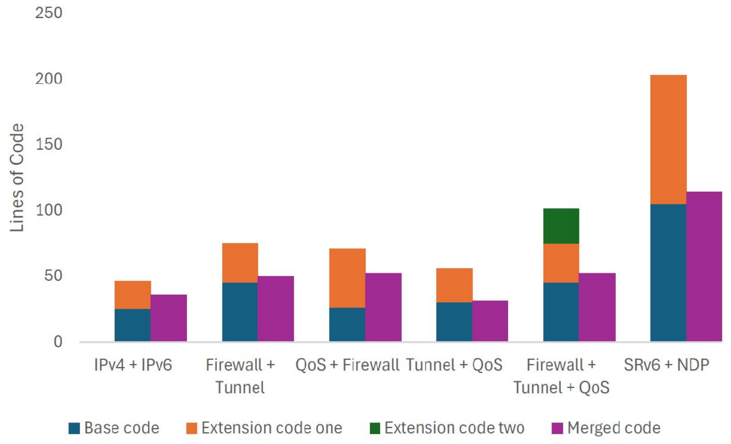
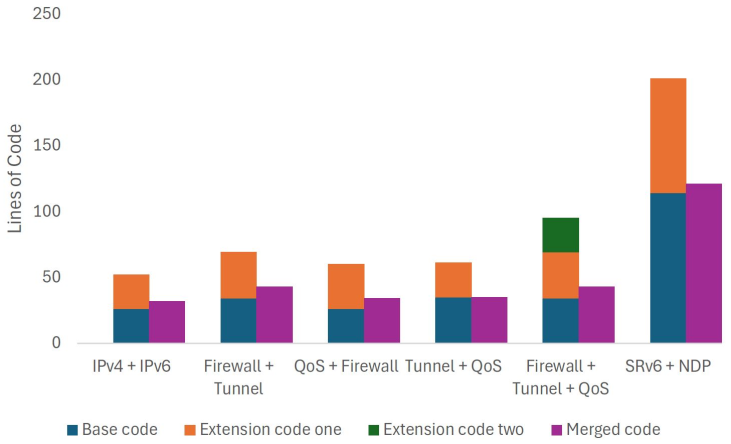


Figure 11. Client-server framework for vendor-customer collaboration [10].

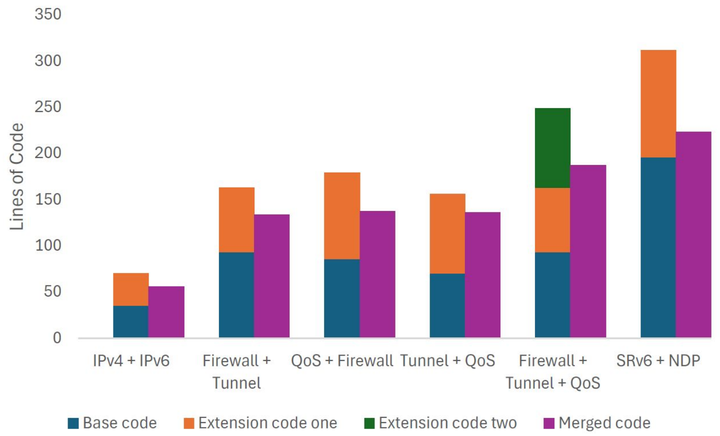
Code Complexity Evaluation



(a) Header and struct components



(b) Parser and deparser components



(c) Ingress and egress component

Figure 12. Comparison of lines of code across components in base, extension, and merged codes [10].

Performance Analysis

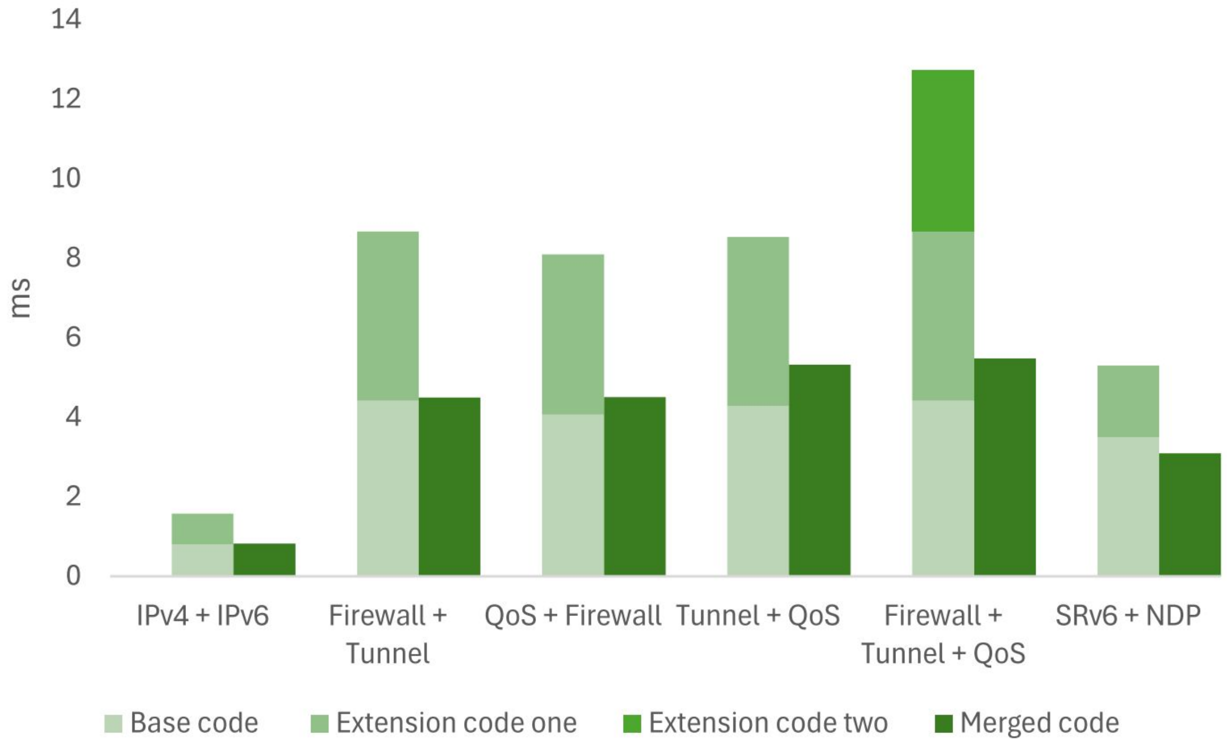


Figure 13. Comparison of Latency Between Base, Extension, and Merged Codes [10].

Performance Analysis

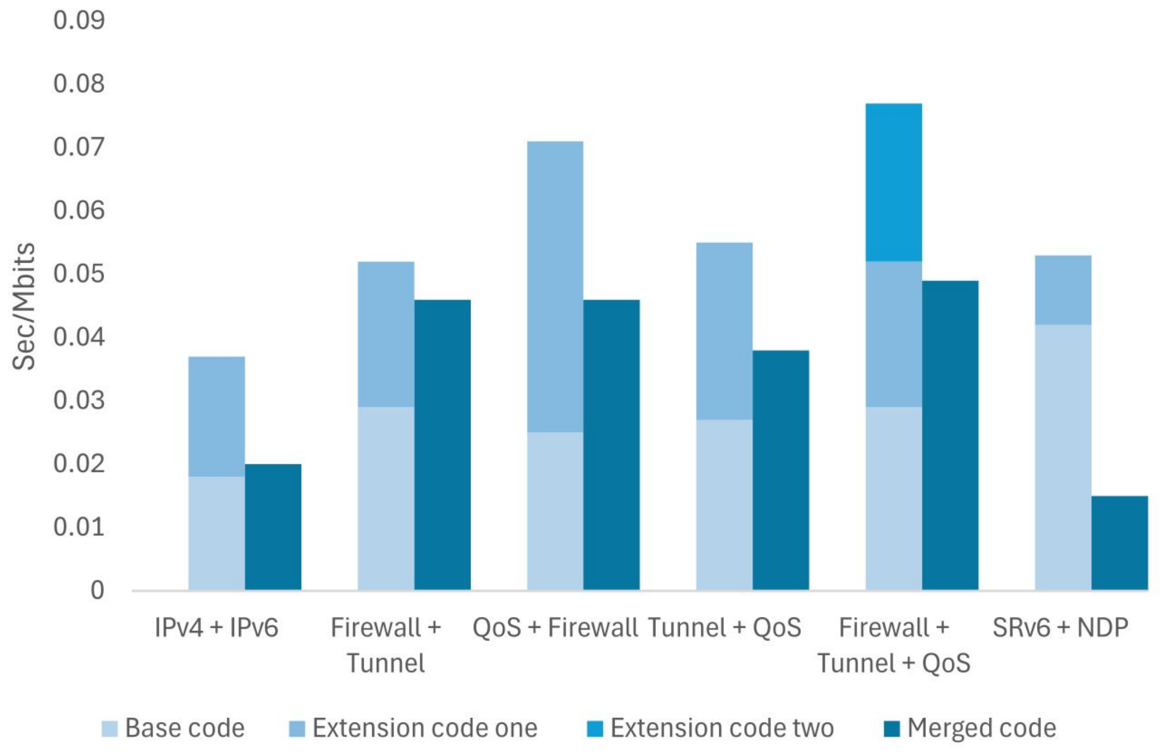
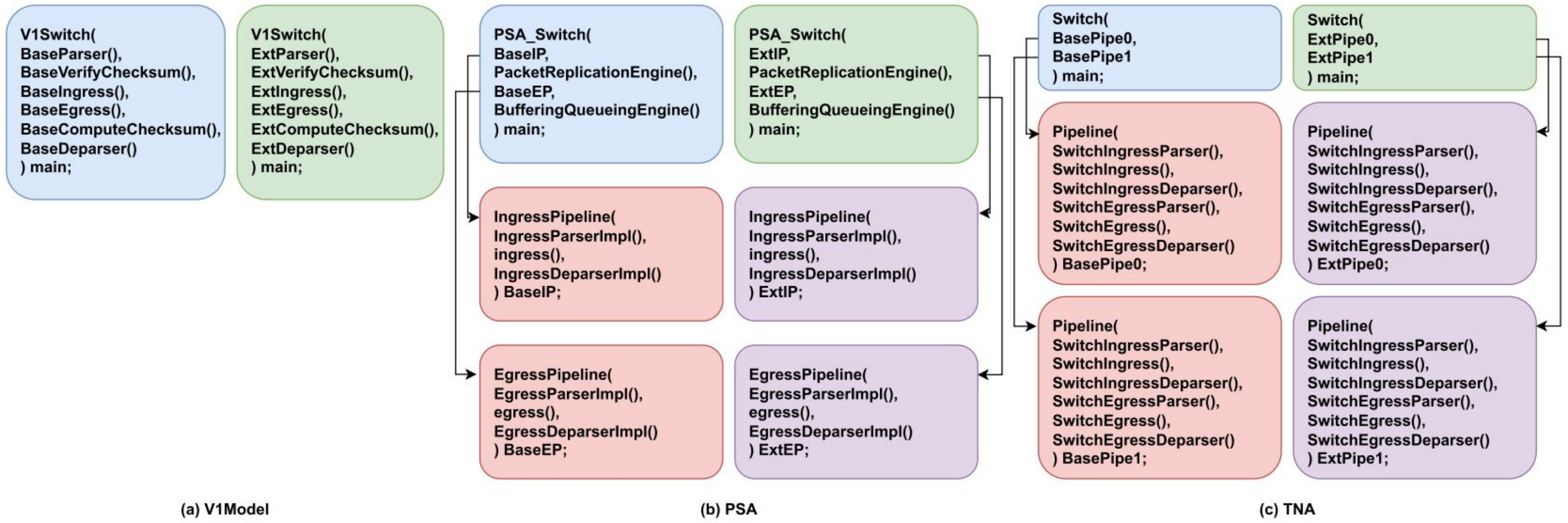


Figure 14. Comparison of Reciprocal of Throughput Between Base, Extension, and Merged Codes [10].

The Design Considerations for Architecture Agnostic Support



```
V1Switch(
BaseParser(),
BaseVerifyChecksum(),
BaseIngress(),
BaseEgress(),
BaseComputeChecksum(),
BaseDeparser()
) main;
```

```
V1Switch(
ExtParser(),
ExtVerifyChecksum(),
ExtIngress(),
ExtEgress(),
ExtComputeChecksum(),
ExtDeparser()
) main;
```

```
PSA_Switch(
BaseIP,
PacketReplicationEngine(),
BaseEP,
BufferingQueueingEngine()
) main;
```

```
PSA_Switch(
ExtIP,
PacketReplicationEngine(),
ExtEP,
BufferingQueueingEngine()
) main;
```

```
IngressPipeline(
IngressParserImpl(),
ingress(),
IngressDeparserImpl()
) BaseIP;
```

```
IngressPipeline(
IngressParserImpl(),
ingress(),
IngressDeparserImpl()
) ExtIP;
```

```
EgressPipeline(
EgressParserImpl(),
egress(),
EgressDeparserImpl()
) BaseEP;
```

```
EgressPipeline(
EgressParserImpl(),
egress(),
EgressDeparserImpl()
) ExtEP;
```

```
Switch(
BasePipe0,
BasePipe1
) main;
```

```
Switch(
ExtPipe0,
ExtPipe1
) main;
```

```
Pipeline(
SwitchIngressParser(),
SwitchIngress(),
SwitchIngressDeparser(),
SwitchEgressParser(),
SwitchEgress(),
SwitchEgressDeparser()
) BasePipe0;
```

```
Pipeline(
SwitchIngressParser(),
SwitchIngress(),
SwitchIngressDeparser(),
SwitchEgressParser(),
SwitchEgress(),
SwitchEgressDeparser()
) ExtPipe0;
```

```
Pipeline(
SwitchIngressParser(),
SwitchIngress(),
SwitchIngressDeparser(),
SwitchEgressParser(),
SwitchEgress(),
SwitchEgressDeparser()
) BasePipe1;
```

```
Pipeline(
SwitchIngressParser(),
SwitchIngress(),
SwitchIngressDeparser(),
SwitchEgressParser(),
SwitchEgress(),
SwitchEgressDeparser()
) ExtPipe1;
```

Figure 15. Modular comparison workflow across V1Model, PSA, and TNA architectures. [10].

Need of Object-oriented System

- RQ2: Can object-oriented style inheritance be added to P4 and give more modularity?

- Previous work only supported modularity at the table level.
- Lack of mechanisms to reuse subcomponents within a table.
- Next study introduces inheritance to enable fine-grained

- M. Rahmati, F.-R. Boyer, B. Pontikakis, J.-P. David, and Y. Savaria, "P4O2: Enabling Object-Oriented-Inspired Modularity in P4," under revision at IEEE Access [11].

Conclusion

- **Key Findings**
  - Modularity reduces LoC by an average of 33% in SRv6 with NDP
  - Improves throughput by up to 44% for example in SRv6 with NDP
  - Enabling multi-program execution in one program in Tunnel + QoS + Firewall

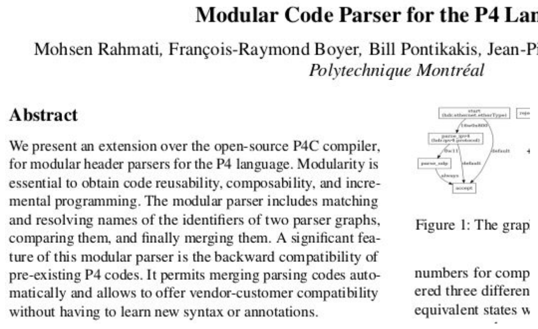
Limitations

- Static Modularity Only
- Evaluation Limited to Reference Architectures
- Tooling and Ecosystem Integration

Future Researches

- Dynamic and Runtime Modularity
- Standardization and Ecosystem Integration
- Modularity Across Control Planes

Publications



Modular Code Parser for the P4 Language

Mohsen Rahmati, François-Raymond Boyer, Bill Pontikakis, Jean-Pierre David, *and* Yvon Savaria
Polytechnique Montréal

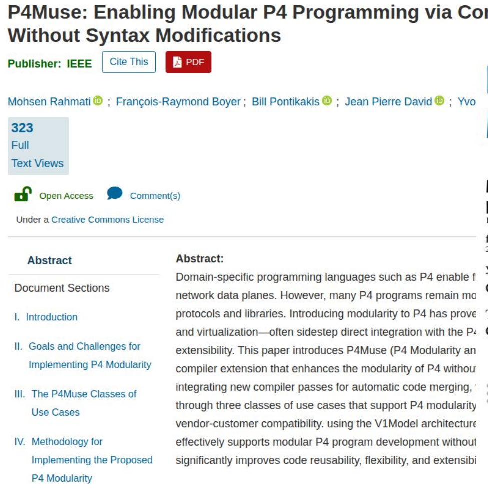
Abstract

We present an extension over the open-source P4C compiler, for modular header parsers for the P4 language. Modularity is essential to obtain code reusability, composability, and incremental programming. The modular parser includes matching and resolving names of the identifiers of two parser graphs, comparing them, and finally merging them. A significant feature of this modular parser is the backward compatibility of pre-existing P4 codes. It permits merging parsing codes automatically and allows to offer vendor-customer compatibility without having to learn new syntax or annotations.

Figure 1: The grap

numbers for comp
ered three differen
equivalent states w

Presented in P4 Workshop, 2023



P4Muse: Enabling Modular P4 Programming via Compiler-Managed Code Merging Without Syntax Modifications

Publisher: IEEE Cite This PDF

Mohsen Rahmati ; François-Raymond Boyer ; Bill Pontikakis ; Jean Pierre David ; Yvo

323 Full Text Views

Open Access Comment(s)

Under a Creative Commons License

Abstract

Document Sections

I. Introduction

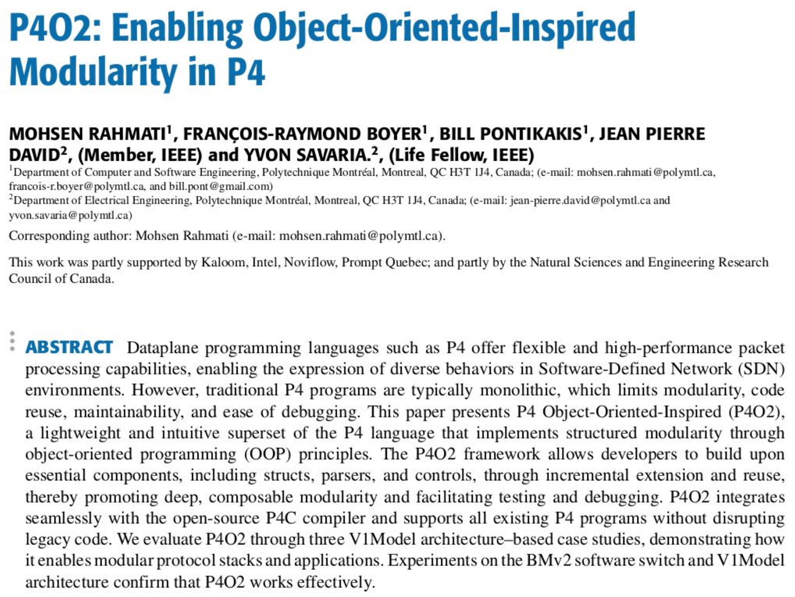
II. Goals and Challenges for Implementing P4 Modularity

III. The P4Muse Classes of Use Cases

IV. Methodology for Implementing the Proposed P4 Modularity

Abstract:
Domain-specific programming languages such as P4 enable fl
network data planes. However, many P4 programs remain mo
protocols and libraries. Introducing modularity to P4 has prove
and virtualization—often sidestep direct integration with the P4
extensibility. This paper introduces P4Muse (P4 Modularity an
compiler extension that enhances the modularity of P4 withou
integrating new compiler passes for automatic code merging,
through three classes of use cases that support P4 modularity
vendor-customer compatibility. using the V1Model architecture
effectively supports modular P4 program development without
significantly improves code reusability, flexibility, and extensibi

Published in IEEE Access, 2025



P4O2: Enabling Object-Oriented-Inspired Modularity in P4

MOHSEN RAHMATI[1], FRANÇOIS-RAYMOND BOYER[1], BILL PONTIKAKIS[1], JEAN PIERRE DAVID[2], (Member, IEEE) and YVON SAVARIA.[2], (Life Fellow, IEEE)
[1]Department of Computer and Software Engineering, Polytechnique Montréal, Montreal, QC H3T 1J4, Canada; (e-mail: mohsen.rahmati@polymtl.ca, francois-r.boyer@polymtl.ca, and bill.pont@gmail.com)
[2]Department of Electrical Engineering, Polytechnique Montréal, Montreal, QC H3T 1J4, Canada; (e-mail: jean-pierre.david@polymtl.ca and yvon.savaria@polymtl.ca)
Corresponding author: Mohsen Rahmati (e-mail: mohsen.rahmati@polymtl.ca).

This work was partly supported by Kaloom, Intel, Noviflow, Prompt Quebec; and partly by the Natural Sciences and Engineering Research Council of Canada.

ABSTRACT Dataplane programming languages such as P4 offer flexible and high-performance packet processing capabilities, enabling the expression of diverse behaviors in Software-Defined Network (SDN) environments. However, traditional P4 programs are typically monolithic, which limits modularity, code reuse, maintainability, and ease of debugging. This paper presents P4 Object-Oriented-Inspired (P4O2), a lightweight and intuitive superset of the P4 language that implements structured modularity through object-oriented programming (OOP) principles. The P4O2 framework allows developers to build upon essential components, including structs, parsers, and controls, through incremental extension and reuse, thereby promoting deep, composable modularity and facilitating testing and debugging. P4O2 integrates seamlessly with the open-source P4C compiler and supports all existing P4 programs without disrupting legacy code. We evaluate P4O2 through three V1Model architecture–based case studies, demonstrating how it enables modular protocol stacks and applications. Experiments on the BMv2 software switch and V1Model architecture confirm that P4O2 works effectively.

Under revision of IEEE Access, 2025

References

[1] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, “A survey on data plane programming with P4: Fundamentals, advances, and applied research,” Journal of Network and Computer Applications, vol. 212, p. 103561, 2023.
[2] Bicycle, by dapple-designers. Pixabay. https://pixabay.com/vectors/cycle-bicycle-load-green-bike-4536914/ (Published Oct 10 2019). Used under the Pixabay Content License.
[3] H. Soni, M. Rifai, P. Kumar, R. Doenges, and N. Foster, "Composing Dataplane Programs with uP4," in Proc. ACM SIGCOMM ’20, 2020, pp. 329–343.
[**4**] J. Gao, E. Zhai, H. H. Liu, R. Miao, Y. Zhou, B. Tian, C. Sun, D. Cai, M. Zhang, and M. Yu, "Lyra: A Cross-Platform Language and Compiler for Data Plane Programming on Heterogeneous ASICs," in Proc. ACM SIGCOMM ’20, 2020, pp. 435–450.
[5] M. Baldi. 2019. “daPIPE a Data Plane Incremental Programming Environment,” In 2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS ’19). IEEE, 1–6.
[6] A. Fattaholmanan, M. Baldi, A. Carzaniga, and R. Soulé, "P4 Weaver: Supporting Modular and Incremental Programming in P4,"
in Proc. ACM SIGCOMM Symposium on SDN Research (SOSR ’21), 2021, pp. 54–65.
[7] MNK Lans and Consulting. 2021. p4-ansible. https://mnkcg.com/products/p4-ansible/. Accessed: 2023-03-08.

References

[8] "the open-source p4 compiler". https://github.com/p4lang/p4c.
[9] M. Rahmati, F.-R. Boyer, B. Pontikakis, J.-P. David, and Y. Savaria, "Modular Code Parser for the P4 Language," presented at the P4 Workshop, 2023.
[10] M. Rahmati, F. -R. Boyer, B. Pontikakis, J. Pierre David, and Y. Savaria, "P4Muse: Enabling Modular P4 Programming via Compiler-Managed Code Merging Without Syntax Modifications," in IEEE Access, vol. 13, pp. 124138-124157, 2025.
[11] M. Rahmati, F.-R. Boyer, B. Pontikakis, J.-P. David, and Y. Savaria, "P4O2: Enabling Object-Oriented-Inspired Modularity in P4," under revision at IEEE Access.

T	h	a	n	k	Y	o	u

Q	u	e	s	t	i	o	n	s	?