Script-Based Approach to P4 Data Plane Programming

Conference Presentation: KuVS FG NetSoft 2025

Pascal Bast, Thomas Scheffler

Hochschule für Technik und Wirtschaft Berlin

Information and communication technology

April 3, 2025

Background

Network Programming Challenges

- Programmable networking has transformed network research and development
- P4 provides flexible data plane programming capabilities
- High adoption in research settings, slower in commercial environments
- Challenges for typical network engineers or researchers:
  - Steep learning curve for P4 programming
  - Complex data plane development
  - Need for specialized knowledge of architecture

Problem Statement

Key Challenges in P4 Deployment

- Traditional network engineers often lack P4 programming experience
- Manual adaptation of P4 code is:
  - Time-consuming
  - Error-prone
  - Hard to debug
- Need for simple abstractions
- Particular use case: Network security and testing

Objectives

Our Approach

- Development of a script-based approach for P4 programming
- Focus on usability for security researchers without deep P4 knowledge
- Implementation of a DDoS stresser application on Intel Tofino switch platform
- Template-based generation of P4 code and switch configuration

P4 Programming Model

Data Plane Programming

- P4 code defines the data plane as:
  - Headers and their structure
  - Parser state machine
  - Control blocks with match-action tables
  - Actions to be performed on packets
- Control plane provides runtime configuration:
  - Table entries
  - Action parameters
  - Runtime metrics

Template-Based Approach

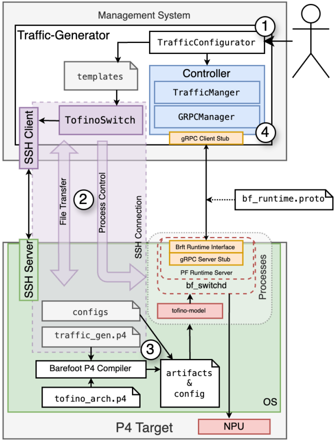
Hybrid Configuration Model

- Structural parameters (affecting packet processing logic):
  - Protocol header definitions
  - Parser states
  - Match-action pipeline structures
  - Requires template-based P4 code generation
- Runtime parameters (configurable during operation):
  - IP ranges, MAC addresses
  - Port configurations
  - Traffic flow specifications
  - Configured via data plane API

Architecture and System Workflow

Operational Process

1. User defines test parameters in Python script
2. Generated code and configs are transferred to switch
3. Compilation and deployment on target
4. Runtime configuration via data plane API (BF Runtime)



Code Generation Example

Python Configuration

```python
traffic_conf.add_packet_data(
  source_cidr="192.168.178.1/24",
  destination_cidr="192.168.178.25/32",
  protocols=["tcp", "ipv4"],
  pkt_len=500
)
```

Generated P4 Parser (Jinja Template)

```
state parse_ethernet {
  pkt.extract(hdr.ethernet);
  transition select(hdr.ethernet.etherType)
  {
    {% if 'ipv4' in protocols %}
    TYPE_IPV4: parse_ipv4;
    {% else %}
    default: accept;
    {% endif %}
  }
}
```

Implementation Details

Code Generation

- Jinja templates for P4 code
- Dynamic generation of:
  - Header definitions
  - Parser states
  - Control blocks
  - Match-action tables
- Configuration files via templates

Switch Configuration

```
# Port configuration settings
traffic_conf.add_physical_output_port(
  output_physical_port=1,
  port_speed="25G"
)

# Throughput settings
traffic_conf.add_throughput(
  throughput_mbps=1000,
  mode="port_shaping"
)
```

Integration with Data Plane API

Barefoot Runtime Interface

- Python bindings for BF Runtime gRPC interface
- Programmatic access to:
  - Table configuration
  - Action parameters
  - Statistics and counters
- Real-time monitoring capabilities
- Unified interface for most switch operations

DDoS Stresser Implementation

Application Capabilities

- Generation of high-flow packet streams
- Support for common DDoS attack patterns:
  - TCP SYN floods
  - UDP floods
  - ICMP ping floods
- Configurable parameters:
  - Source/destination IP ranges
  - Packet size and protocol
  - Traffic rate and duration

Key Findings

Benefits of Script-Based Approach

- Simplified Configuration:
  - High-level abstractions hide P4 complexity
  - Consistent code generation across deployments
  - Reduced human error in programming
- Improved Usability:
  - Network security personnel can operate without P4 knowledge
  - Quick modifications without deep switch understanding
  - Repeatable test scenarios

Performance

System Efficiency

- Initial setup time dominated by P4 compilation
- Subsequent configuration changes execute within seconds
- High packet generation rate (limited only by hardware)
- Ability to saturate 25G/100G interfaces
- Scriptable test sequences for complex scenarios

Future Work

Development Opportunities

- Integration with Open Traffic Generator API standard
- Extension to support more complex traffic patterns
- Additional protocol support
- Integration with automation frameworks

Thank You

Contact Information

Pascal Bast (<pascal.bast@googlemail.com>)
Thomas Scheffler (<thomas.scheffler@htw-berlin.de>)

GitHub: <https://github.com/pascalb97/tofino-traffic-generator>