Runtime Programmable Switches

Jiarong Xing, Kuo-Feng Hsu, Matty Kadosh, Alan Lo, Yonatan Piasetzky, Arvind Krishnamurthy, Ang Chen
Background: Programmable switches

- Switches become fixed after the program is deployed.
- Operators compile and reflash the data plane w/ a new program.
- However, reflashing can cause downtime and packet loss.
- To avoid downtime, traffic drain/undrain is necessary:
  - Changes must be infrequent and operator driven.
From compile-time to runtime programmability

The key features of runtime programmability:

- **Runtime**: Live upgrade at runtime
- **Seamless**: Zero downtime and packet loss
- **Partial**: Upgrade the program partially
- **Atomic**: Reprogram with strong consistency guarantees
Benefits of runtime programmability

- Example benefits of runtime programmability
  - Real-time attack mitigation
  - Just-in-time network optimization
  - Tenant-specific network extension
An ecosystem that supports **live** program upgrades with **strong consistency** and **no downtime**.
Challenge #1: Flexible switch architecture

- The RMT architecture is inflexible for runtime changes
  - Compute and memory are tightly coupled in stages.
  - Packets can only move forward to the next stage.
  - Memory of one stage cannot be used by other stages.

Diagram showing the architecture with stages and memory allocation.
Solution: Disaggregated RMT architecture

• Enhanced disaggregated RMT (dRMT) architecture
  • Compute and memory are disaggregated
  • Memory is sharded, and accesses are load-balanced
  • MA processors handle packets in parallel in a run-to-completion manner
Partial reconfiguration with indirection

- FlexCore adopts a pointer-based indirection mechanism
- Program description table (PDTab)
  - Each PDTab entry records the information of a MA table
  - PDTab entries are chained together by “next table pointers”
  - Pointers can be changed at runtime atomically.

Program description table (PDTab)

Memory banks

P4 program

```p4
parser {
...
}

control ingress{
  ipv4.apply();
  acl.apply();
  route.apply();
}
```
More partial reconfiguration primitives

- `AllocTab(T)`
- `SetPtr(T, T')`
- `ModTab(T, T')`

- `AllocCond(B, Pred, Br1, Br2)`
- `DeallocCond(B)`
- `SetCondPtr(B, N1, N2)`

- `AllocState(S)`
- `AllocTrans(S1, S2)`
- `AllocEx(R)`

* Each allocation primitive has its respective deallocation primitive

- FlexCore provides a set of atomic reconfiguration primitives
- FlexCore transforms program diff into these primitives
Challenge #3: Atomic changes

- A program-level update may involve multiple discrete changes
- Non-atomic changes lead to undesired intermediate states
Solution: Version control with FlexEdge

- **FlexEdge**: A version control mechanism
  - Check on a global version metadata
  - Can be inserted/deleted one by one atomically
Challenge #4: Finer-grained partial updates

- One tx still requires preparing all the differences together
- Could fail if the switch has insufficient headroom
- First completed can release resources for later updates
- We need multi-step transactions
Solution: Multi-level consistency for multi-step TXs

1. **FlexCore supports multi-level consistency**
2. **Definitions and algorithms are in the paper!**

Weaker consistency, lower transient overhead
Implementation and setup

• Commercial switch hardware
  • NVIDIA/Mellanox Spectrum-2 silicon
  • As fast as 12.8Tbps

• BMv2 emulator
  • Reconfiguration primitives in P4Runtime
  • Three consistency levels
  • Available at https://github.com/jiarong0907/FlexCore

• Case study setup
  • Accelerated multicast
  • A spectrum-2 switch connected with one sender and several subscribers
  • The sender sends the same data to its subscribers
Case study: Accelerated multicast

- Runtime switch function upgrade with FlexCore has no downtime
- Runtime network optimization greatly improves performance

Traffic throughput

Zero packet loss
20ns reduced

Throughput (Gbps)

Time (sec)

Unicast
Insert multicast
Insert telemetry
Remove multicast
Remove telemetry

Time

Num of subscribers

Unicast
Elmo

Time completion

Job completion time

100
125

Time

20ns reduced
More results in the paper

Use cases

- In-place application upgrade
- Real-time attack mitigation
- Tenant-specific network extensions

Multi-level consistency

- Scalability of real-world P4 programs
- Large-scale synthetic program simulation

System overhead

- Hardware overhead
- Reconfiguration transient overhead
Summary

• Today’s switches are only programmable at compile time
• FlexCore: An ecosystem for runtime programmability
  • Live switch program upgrades
  • Zero packet loss, no downtime
  • Support partial upgrades with multi-level consistency guarantees
• Implementation:
  • 12.8Tbps Spectrum-2 switch silicon
  • BMv2 emulator: https://github.com/jiarong0907/FlexCore
• Use cases:
  • Runtime accelerated multicast, real-time attack mitigation, ...
• Ultimate vision:
  • End-to-end runtime programmable networks
  • See our vision paper at HotNets’21: A Vision for Runtime Programmable Networks