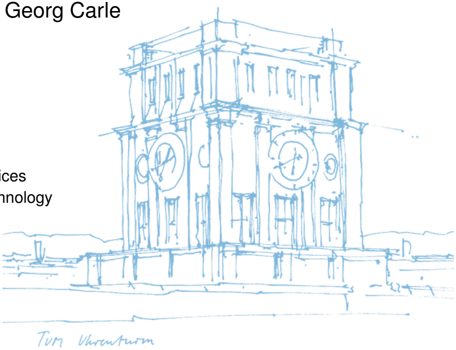
Dynamic Data Plane Updates using Lua and libmoon

Manuel Simon, Sebastian Gallenmüller, Georg Carle

Thursday 3rd April, 2025

KuVS FG NetSoft 2025

Chair of Network Architectures and Services
School of Computation, Information and Technology
Technical University of Munich



Introduction

Motivation

- Modern communication networks have to offer high-performant and reliable connections
- Interrupt-free, dynamic data plane updates increase network resilience
  - application migration (e.g., for failovers)
  - tenant-specific processing
- Just-in-time (JIT) compiled languages seem to be a promising candidates for on-the-fly function updates

Contribution

- LuaJIT/libmoon-based prototype implementation for dynamic network functions
- Investigation of applicability and performance consequences

Background

libmoon

- Lua(JIT)-based wrapper for DPDK
- Allow flexible, high-level, but high-performant packet processing

DPDK

- High-performance packet processing framework
- Bypassing Linux networking kernel stack

Previous and Related Work

Active Networking

- **Capsule-based active networking** [6]: Capsules/packets carry their "own" program fragments
- **Tiny packet programs (TTPs)** [4]: active packets with very restricted number of instructions

P4

- **Active RMT** [1]: Instruction set in P4 allowing changegable functionality
- **FlexCore** [7]: Runtime partial reprogrammable switch architecture
- **In-situ Programmable Data Plane** [3]: Switch architecture and reconfigurable P4 (rP4) for runtime updates

P4/eBPF

- **Dynamic eBPF in P4 pipeline** [5]: Runtime-updatable eBPF processors within P4 pipeline

---

[6] D. L. Tennenhouse and D. J. Wetherall. Towards an Active Network Architecture, *SIGCOMM Comput. Commun. Rev., 26, apr 1996*

Previous and Related Work

Active Networking

- **Capsule-based active networking** [6]: Capsules/packets carry their "own" program fragments
- **Tiny packet programs (TTPs)** [4]: active packets with very restricted number of instructions

P4

- **Active RMT** [1]: Instruction set in P4 allowing changegable functionality
- **FlexCore** [7]: Runtime partial reprogrammable switch architecture
- **In-situ Programmable Data Plane** [3]: Switch architecture and reconfigurable P4 (rP4) for runtime updates

P4/eBPF

- **Dynamic eBPF in P4 pipeline** [5]: Runtime-updatable eBPF processors within P4 pipeline

---

[4] V. Jeyakumar, M. Alizadeh, Y. Geng, C. Kim, and D. Mazières. Millions of Little Minions: Using Packets for Low Latency Network Programming and Visibility., *ACM SIGCOMM 2014*

Previous and Related Work

Active Networking

- **Capsule-based active networking** [6]: Capsules/packets carry their "own" program fragments
- **Tiny packet programs (TTPs)** [4]: active packets with very restricted number of instructions

P4

- **Active RMT** [1]: Instruction set in P4 allowing changegable functionality
- **FlexCore** [7]: Runtime partial reprogrammable switch architecture
- **In-situ Programmable Data Plane** [3]: Switch architecture and reconfigurable P4 (rP4) for runtime updates

P4/eBPF

- **Dynamic eBPF in P4 pipeline** [5]: Runtime-updatable eBPF processors within P4 pipeline

[1] R. Das and A. C. Snoeren. Memory management in Active RMT: Towards Runtime-Programmable Switches, *ACM SIGCOMM 2023*

Previous and Related Work

Active Networking

- **Capsule-based active networking** [6]: Capsules/packets carry their "own" program fragments
- **Tiny packet programs (TTPs)** [4]: active packets with very restricted number of instructions

P4

- **Active RMT** [1]: Instruction set in P4 allowing changegable functionality
- **FlexCore** [7]: Runtime partial reprogrammable switch architecture
- **In-situ Programmable Data Plane** [3]: Switch architecture and reconfigurable P4 (rP4) for runtime updates

P4/eBPF

- **Dynamic eBPF in P4 pipeline** [5]: Runtime-updatable eBPF processors within P4 pipeline

---

[7] J. Xing, K.-F. Hsu, M. Kadosh, A. Lo, Y. Piasetzky, A. Krishnamurthy, and A. Chen. Runtime Programmable Switches, *NSDI 2022*

Previous and Related Work

Active Networking

- **Capsule-based active networking** [6]: Capsules/packets carry their "own" program fragments
- **Tiny packet programs (TTPs)** [4]: active packets with very restricted number of instructions

P4

- **Active RMT** [1]: Instruction set in P4 allowing changegable functionality
- **FlexCore** [7]: Runtime partial reprogrammable switch architecture
- **In-situ Programmable Data Plane** [3]: Switch architecture and reconfigurable P4 (rP4) for runtime updates

P4/eBPF

- **Dynamic eBPF in P4 pipeline** [5]: Runtime-updatable eBPF processors within P4 pipeline

---

[3] Y. Feng, Z. Chen, H. Song, W. Xu, J. Li, Z. Zhang, T. Yun, Y. Wan, and B. Liu. Enabling In-situ Programmability in Network Data Plane: From Architecture to Language, *NSDI 2022*

Previous and Related Work

Active Networking

- **Capsule-based active networking** [6]: Capsules/packets carry their "own" program fragments
- **Tiny packet programs (TTPs)** [4]: active packets with very restricted number of instructions

P4

- **Active RMT** [1]: Instruction set in P4 allowing changegable functionality
- **FlexCore** [7]: Runtime partial reprogrammable switch architecture
- **In-situ Programmable Data Plane** [3]: Switch architecture and reconfigurable P4 (rP4) for runtime updates

P4/eBPF

- **Dynamic eBPF in P4 pipeline** [5]: Runtime-updatable eBPF processors within P4 pipeline

---

[5] M. Simon, H. Stubbe, S. Gallenmüller, and G. Carle. Honey for the Ice Bear - Dynamic eBPF in P4, *eBPF Workshop @ SIGCOMM 2024*

Implementation

Prototype

- Implemented in *libmoon* using LuaJIT

Flow Table

- Every flow has its own function
- Hashtable mapping flows to the (network) function

Function Update

- Lua's built-in `loadstring()` function returns pointer for given source code
- LuaJIT can JIT compile the code
- Several JIT optimization schemes possible (`-O0`, `-O1`, `-O2`, `-O3`)

Implementation

INC ①

payload: 0

DYN ①

```
local pkt = buf:getUDP4IncPacket();
pkt.inc:setPayload(54321);
```

INC ②

payload: 0

INC ①

payload: 0

Flow	Function pointer
Flow ①	`forward`
Flow ②	`forward`
...	...
default	`forward`

```
forward():
local pkt = buf:getUDP4IncPacket();
pkt.inc:setPayload(12345);
```

Implementation

INC ①

payload: 0

DYN ①

```
local pkt = buf:getUDP4IncPacket();
pkt.inc:setPayload(54321);
```

INC ②

payload: 0

INC ①

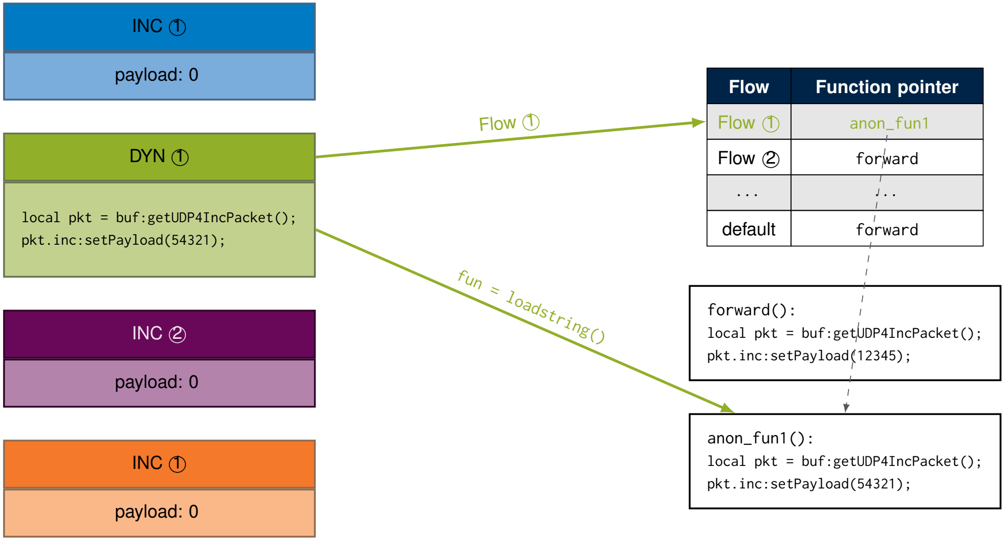
payload: 0

Flow ①

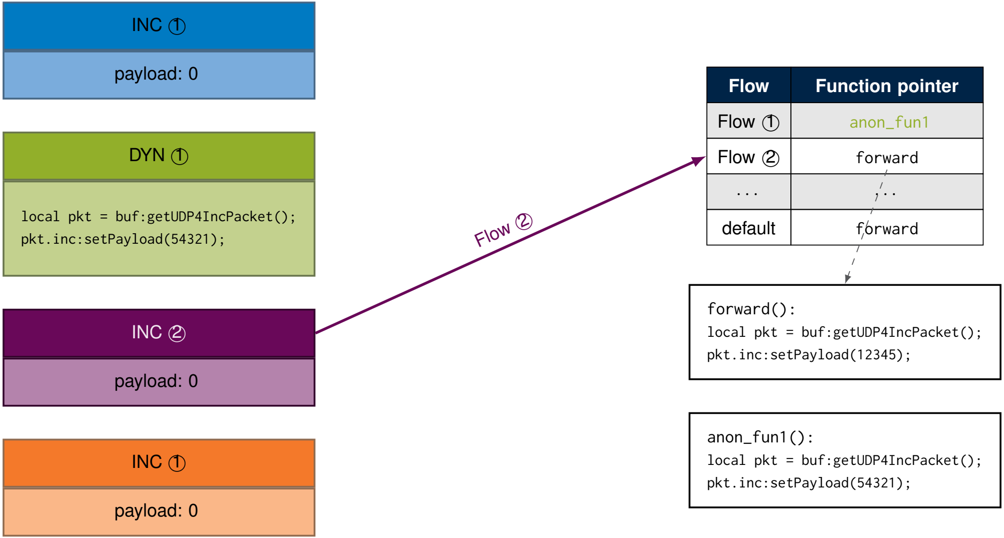
Flow	Function pointer
Flow ①	`forward`
Flow ②	`forward`
...	...
default	`forward`

```
forward():
local pkt = buf:getUDP4IncPacket();
pkt.inc:setPayload(12345);
```

Implementation



Implementation



Implementation

INC ①

payload: 0

DYN ①

```
local pkt = buf:getUDP4IncPacket();
pkt.inc:setPayload(54321);
```

INC ②

payload: 0

INC ①

payload: 0

Flow ①

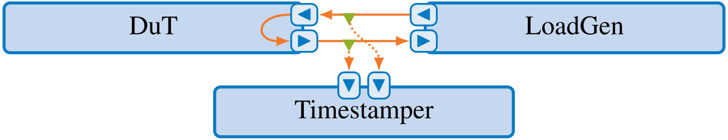
Flow	Function pointer
Flow ①	anon_fun1
Flow ②	forward
...	...
default	forward

```
forward():
local pkt = buf:getUDP4IncPacket();
pkt.inc:setPayload(12345);
```

```
anon_fun1():
local pkt = buf:getUDP4IncPacket();
pkt.inc:setPayload(54321);
```

Measurement Setup

Setup



DuT

- Intel Xeon D-1518 2.2 GHz, 32 RAM
- libmoon with batch size of one

LoadGen

- MoonGen [2] is used to generate traffic
- Packet size 200 B

Timestamper

- Packet streams duplicated using optical splitter
- Timestamps each packet incoming packet
- Resolution: 12,5 ns

Measurement Setup

Procedure

Two flows

- For *flow 1*, the function will be changed during runtime
- For *flow 2*, the function remains unaffected

Procedure

- First, 50 000 INC packets are sent → default/forwarding function
- Then, one DYN packet updates the code for *flow 1*
- Afterward, another 200 000 INC packets are sent and processed

Network Function

- Default function: Set a constant in a specific header field
- Changed function: sets another constant

⇒ minimum possible overhead

Measurement Setup

Methodology

We want to answer the following questions:

1. **What is the overhead when changing the network function during runtime?**
   - $\rightarrow$ DYN packet

Measurement Setup

Methodology

We want to answer the following questions:

1. **What is the overhead when changing the network function during runtime?**
   → DYN packet
2. **How does the change affect other flows and CPU cores (cross-flow and cross-core dependencies)**
   → using *one* or *two* threads/tasks/cores on the DuT

Measurement Setup

Methodology

We want to answer the following questions:

1. **What is the overhead when changing the network function during runtime?**
   - $\rightarrow$ DYN packet
2. **How does the change affect other flows and CPU cores (cross-flow and cross-core dependencies)**
   - $\rightarrow$ using *one* or *two* threads/tasks/cores on the DuT
3. **How does JIT compilation influence the performance during and after changing the code?**
   - $\rightarrow$ enabling/disable LuaJIT

Measurement Setup

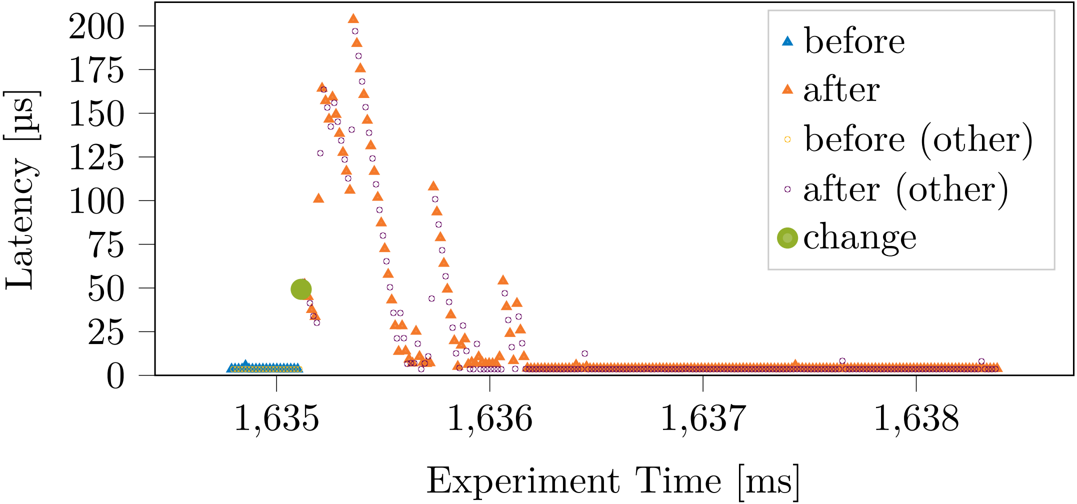
Methodology

We want to answer the following questions:

1. **What is the overhead when changing the network function during runtime?**
   → DYN packet
2. **How does the change affect other flows and CPU cores (cross-flow and cross-core dependencies)**
   → using *one* or *two* threads/tasks/cores on the DuT
3. **How does JIT compilation influence the performance during and after changing the code?**
   → enabling/disable LuaJIT
4. **What are the reasons for performance changes?**
   → manipulating the default function

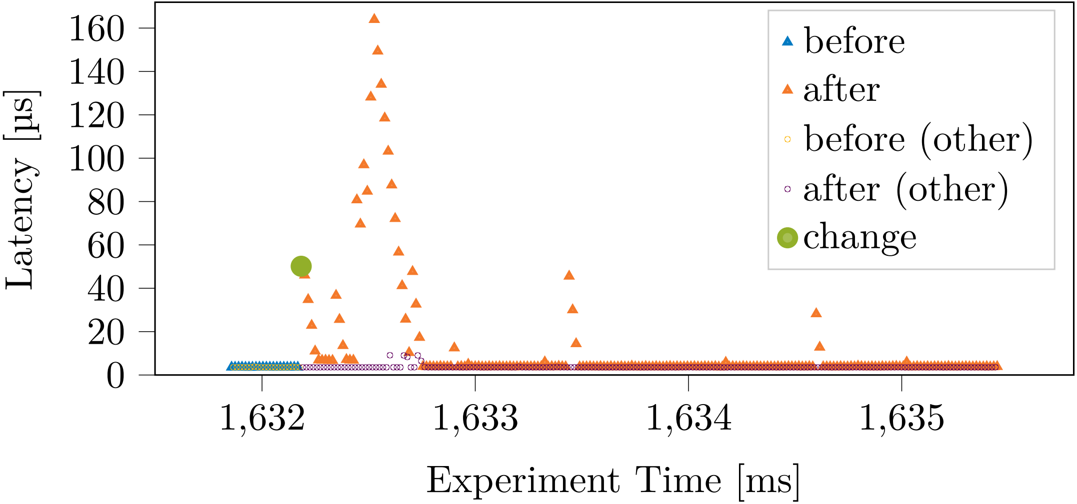
Evaluation

Dynamic program change (one task) (zoomed)



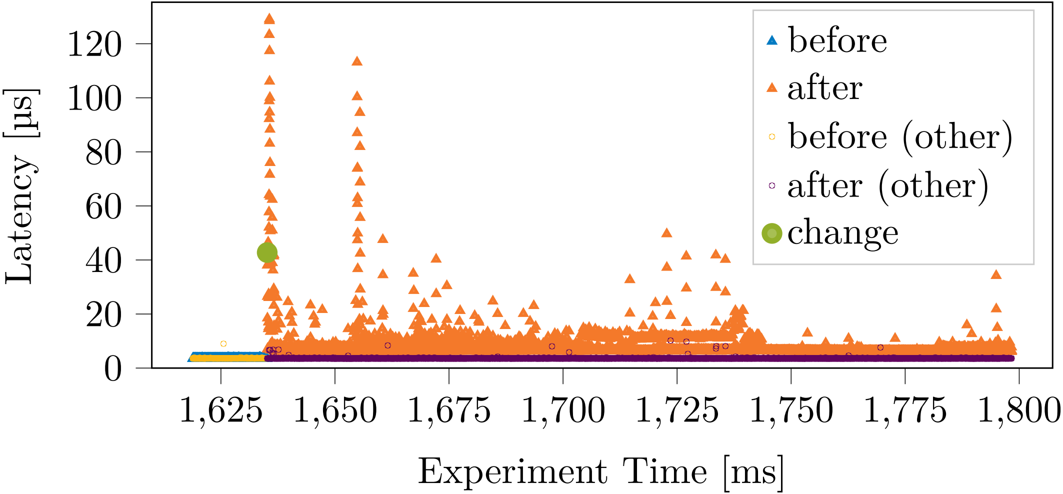
Evaluation

Dynamic program change (two tasks) (zoomed)



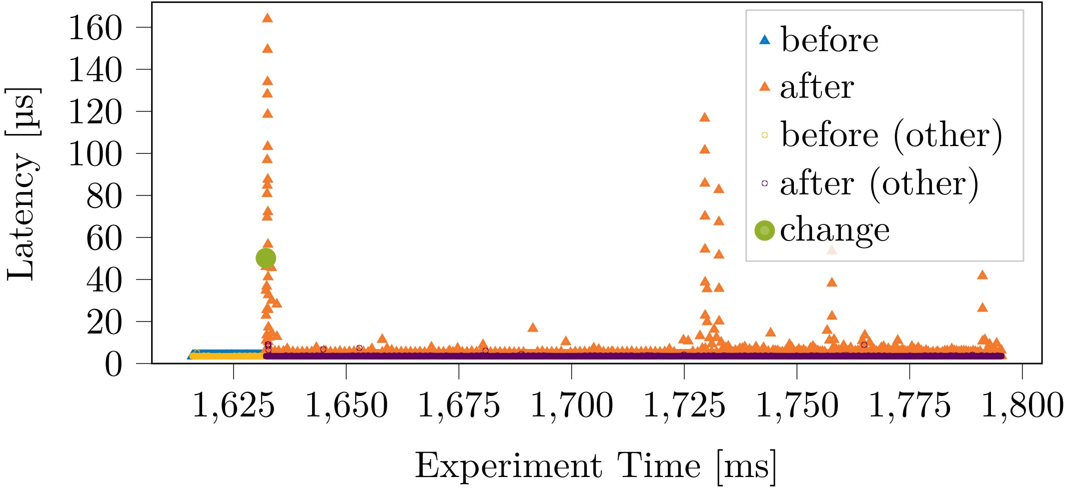
Evaluation

Dynamic program change (two tasks) (**no** JIT)



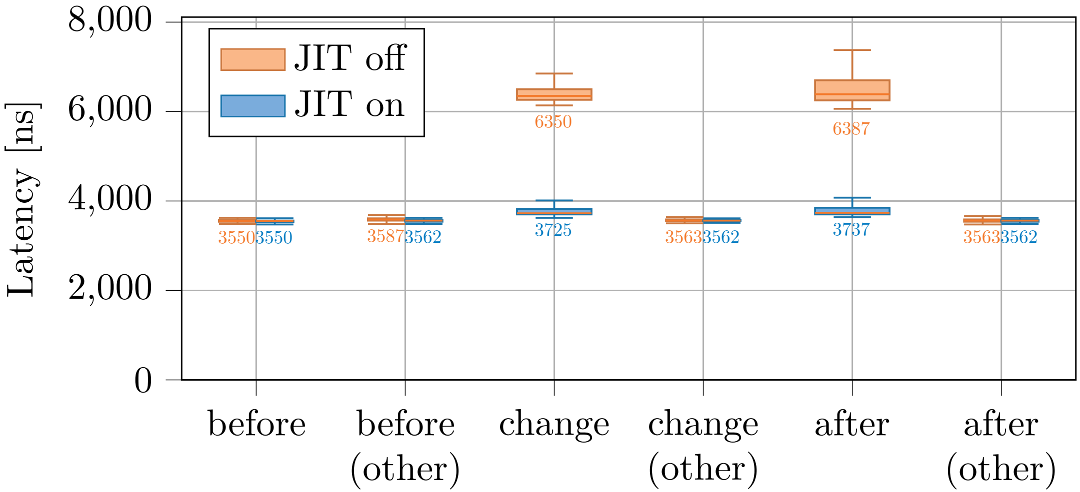
Evaluation

Dynamic program change (two tasks)



Evaluation

Latencies *before*, 5000 packets after the *change*, and there*after*)



Conclusion

Results

- It is feasible to perform dynamic changes with uncompiled source code
- Overhead only for flows processed on the same core
- JIT improves long-term performance, adds minimal overhead to the exchange itself
- Function pointer returned by `loadstring()` adds performance overhead

Future Work

- Investigate different programs (not only baseline overhead)
- Analyze the influence of JIT settings
- Compare to other implementations, e.g., eBPF, XDP
- Investigate the offloading potential such dynamic function to SmartNICs

Bibliography

[1] R. Das and A. C. Snoeren.
Memory Management in ActiveRMT: Towards Runtime-Programmable Switches.
In *ACM SIGCOMM 2023*, page 1043–1059.

[2] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle.
MoonGen: A Scriptable High-Speed Packet Generator.
In *IMC 2015*.

[3] Y. Feng, Z. Chen, H. Song, W. Xu, J. Li, Z. Zhang, T. Yun, Y. Wan, and B. Liu.
Enabling In-situ Programmability in Network Data Plane: From Architecture to Language.
In *NSDI 2022*, pages 635–649.

[4] V. Jeyakumar, M. Alizadeh, Y. Geng, C. Kim, and D. Mazières.
Millions of Little Minions: Using Packets for Low Latency Network Programming and Visibility.
In *ACM SIGCOMM 2014*, page 3–14.

[5] M. Simon, H. Stubbe, S. Gallenmüller, and G. Carle.
Honey for the Ice Bear - Dynamic eBPF in P4.
In *Proceedings of the ACM SIGCOMM 2024 Workshop on eBPF and Kernel Extensions, eBPF 2024, Sydney, NSW, Australia, August 4-8, 2024*. ACM, 2024.

[6] D. L. Tennenhouse and D. J. Wetherall.
Towards an Active Network Architecture.
SIGCOMM Comput. Commun. Rev., 26(2):5–17, apr 1996.

[7] J. Xing, K.-F. Hsu, M. Kadosh, A. Lo, Y. Piasetzky, A. Krishnamurthy, and A. Chen.
Runtime Programmable Switches.
In *NSDI 2022*, pages 651–665.