Optimizing Data Plane Programs for the Network

Johannes Krude[1], Matthias Eichholz[2],
Maximilian Winck[1],
Klaus Wehrle[1], Mira Mezini[2]

[1]RWTH Aachen University, [2]Technische Universität Darmstadt

https://comsys.rwth-aachen.de/

NetPL '19, 2019-08-23

Data Plane Programs

- **Shift from fixed to programmable data planes**
  - P4 on e.g., Barefoot Tofino
  - eBPF on NPUs, e.g., Netronome NFP-6000
  - eBPF with XDP e.g., Intel Core-i7
- **Programs are executed for each packet**
- **Limited per-packet time budget**
  - e.g., 67.2 ns at 10GbE
- **Limited program size**
  - NFP-6000: 8k instructions, XDP: 4k instructions, Tofino: 10-20 stages

Data Plane Programs

- **Shift from fixed to programmable data planes**
  - P4 on e.g., Barefoot Tofino
  - eBPF on NPUs, e.g., Netronome NFP-6000
  - eBPF with XDP e.g., Intel Core-i7
- **Programs are executed for each packet**
- **Limited per-packet time budget**
  - e.g., 67.2 ns at 10GbE
- **Limited program size**
  - NFP-6000: 8k instructions, XDP: 4k instructions, Tofino: 10-20 stages
- **Existing optimization work**
  - Optimizing Flow Rules in SDN [Kang et.al., CoNEXT '13]
  - Optimizing programs on individual network nodes [P5, SOSR '17] [Rétvári et.al., NetPL '17]

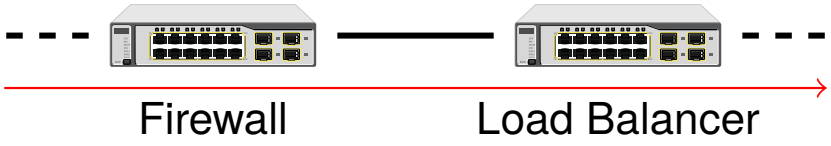
Data Plane Programs

- **Shift from fixed to programmable data planes**
  - P4 on e.g., Barefoot Tofino
  - eBPF on NPUs, e.g., Netronome NFP-6000
  - eBPF with XDP e.g., Intel Core-i7
- **Programs are executed for each packet**
- **Limited per-packet time budget**
  - e.g., 67.2 ns at 10GbE
- **Limited program size**
  - NFP-6000: 8k instructions, XDP: 4k instructions, Tofino: 10-20 stages
- **Existing optimization work**
  - Optimizing Flow Rules in SDN [Kang et.al., CoNEXT '13]
  - Optimizing programs on individual network nodes [P5, SOSR '17] [Rétvári et.al., NetPL '17]

Our proposal

- Integrating the network topology into program optimization

Optimizing for the Network



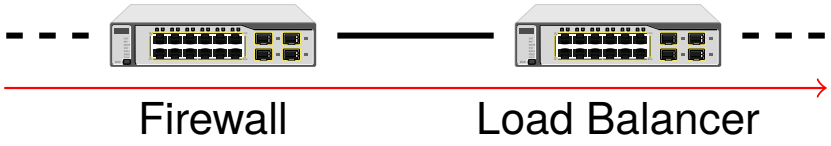
Firewall

- Drops non IPv6 TCP
- Inserts fixed sized IPv6 options
  - e.g., Authentication Header

cilium.io Load Balancer

1. Parse Ethernet header
2. Parse IPv6 header
3. Loop: Parse IPv6 option
4. Extract IPv6 dst & TCP dst
5. Lookup & redirect to backend

Optimizing for the Network



- **Packets to the Load Balancer have the following properties**
  - IPv6 TCP with some IPv6 options
  - IPv6 & TCP dst have a fixed offset

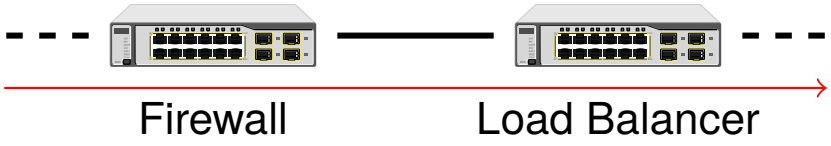
Firewall

- Drops non IPv6 TCP
- Inserts fixed sized IPv6 options
  - e.g., Authentication Header

cilium.io Load Balancer

1. Parse Ethernet header
2. Parse IPv6 header
3. Loop: Parse IPv6 option
4. Extract IPv6 dst & TCP dst
5. Lookup & redirect to backend

Optimizing for the Network



- **Packets to the Load Balancer have the following properties**
  - IPv6 TCP with some IPv6 options
  - IPv6 & TCP dst have a fixed offset
- **The parser has a fixed result**
  - Always the same headers
  - At always the same offsets

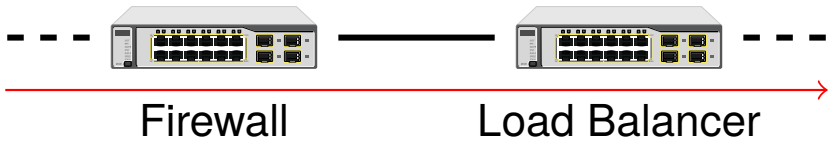
Firewall

- Drops non IPv6 TCP
- Inserts fixed sized IPv6 options
  - e.g., Authentication Header

cilium.io Load Balancer

1. Parse Ethernet header
2. Parse IPv6 header
3. Loop: Parse IPv6 option
4. Extract IPv6 dst & TCP dst
5. Lookup & redirect to backend

Optimizing for the Network



- **Packets to the Load Balancer have the following properties**
  - IPv6 TCP with some IPv6 options
  - IPv6 & TCP dst have a fixed offset
- **The parser has a fixed result**
  - Always the same headers
  - At always the same offsets

Parts of a data plane program are redundant if equivalent functionality was already applied to packets before reaching this program.

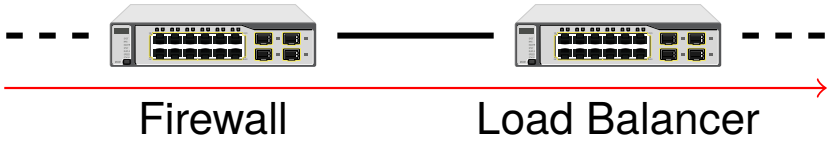
Firewall

- Drops non IPv6 TCP
- Inserts fixed sized IPv6 options
  - e.g., Authentication Header

cilium.io Load Balancer

1. Parse Ethernet header
2. Parse IPv6 header
3. Loop: Parse IPv6 option
4. Extract IPv6 dst & TCP dst
5. Lookup & redirect to backend

Removing Redundancies



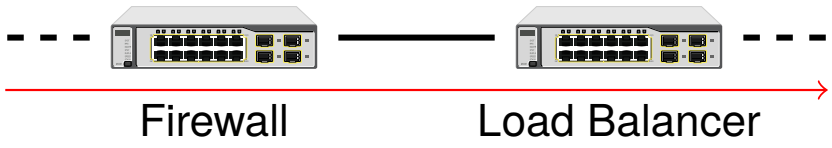
Firewall

- Drops non IPv6 TCP
- Inserts fixed sized IPv6 options
  - e.g., Authentication Header

cilium.io Load Balancer

1. ~~Parse Ethernet header~~
2. ~~Parse IPv6 header~~
3. ~~Loop: Parse IPv6 option~~
4. Extract IPv6 dst & TCP dst
5. Lookup & redirect to backend

Removing Redundancies



Removing redundancies in a downstream program does not change the behavior.

- **The same results are computed**
- **The same packets are transmitted**

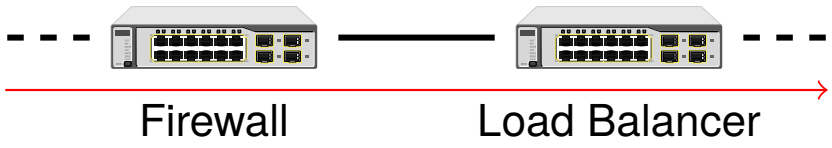
Firewall

- Drops non IPv6 TCP
- Inserts fixed sized IPv6 options
  - e.g., Authentication Header

cilium.io Load Balancer

1. ~~Parse Ethernet header~~
2. ~~Parse IPv6 header~~
3. ~~Loop: Parse IPv6 option~~
4. Extract IPv6 dst & TCP dst
5. Lookup & redirect to backend

Removing Redundancies



Removing redundancies in a downstream program does not change the behavior.

- **The same results are computed**
- **The same packets are transmitted**
- **But, only for this program chain**

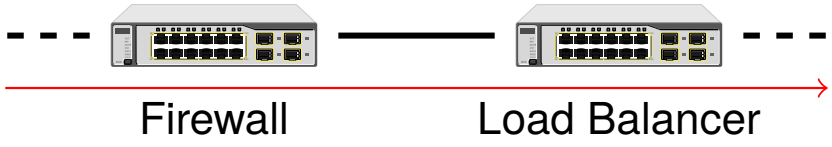
Firewall

- Drops non IPv6 TCP
- Inserts fixed sized IPv6 options
  - e.g., Authentication Header

cilium.io Load Balancer

1. ~~Parse Ethernet header~~
2. ~~Parse IPv6 header~~
3. ~~Loop: Parse IPv6 option~~
4. Extract IPv6 dst & TCP dst
5. Lookup & redirect to backend

Removing Redundancies



Removing redundancies in a downstream program does not change the behavior.

- **The same results are computed**
- **The same packets are transmitted**
- **But, only for this program chain**

This example, XDP BPF Core i7-870

⇒ **2.3**× packet rate (2.5→5.8 M pkt/s)

⇒ **12%** program size(7.6→0.9 KiB)

Firewall

- Drops non IPv6 TCP
- Inserts fixed sized IPv6 options
  - e.g., Authentication Header

cilium.io Load Balancer

1. ~~Parse Ethernet header~~
2. ~~Parse IPv6 header~~
3. ~~Loop: Parse IPv6 option~~
4. Extract IPv6 dst & TCP dst
5. Lookup & redirect to backend

Optimizing Program Chains

Optimizing Program Chains

1. Program combination

```
if (firewall(&packet) != DROP)
    load_balancer(&packet);
```

Optimizing Program Chains

1. Program combination

```
if (firewall(&packet) != DROP)
    load_balancer(&packet);
```

2. Discovering redundancies

- Analyze combined program
- Identify conditions that hold for all executions

Optimizing Program Chains

1. Program combination

```
if (firewall(&packet) != DROP)
    load_balancer(&packet);
```

2. Discovering redundancies

- Analyze combined program
- Identify conditions that hold for all executions

→ Identify dead branches

→ Identify constants in packet

→ …

Optimizing Program Chains

1. Program combination

```
if (firewall(&packet) != DROP)
    load_balancer(&packet);
```

2. Discovering redundancies

- Analyze combined program
- Identify conditions that hold for all executions

→ Identify dead branches

→ Identify constants in packet

→ …

Problem: Conventional tools (e.g., llvm) only output the combined optimized program

Solution: Instead use exhaustive symbolic execution

Optimizing Program Chains

1. Program combination

```
if (firewall(&packet) != DROP)
    load_balancer(&packet);
```

2. Discovering redundancies

- Analyze combined program
- Identify conditions that hold for all executions

→ Identify dead branches
→ Identify constants in packet
→ …

3. Optimization

- Replace instructions in downstream program

→ Replace dead branch by jump
→ Replace constant read by value

Optimizing Program Chains

1. Program combination

```
if (firewall(&packet) != DROP)
    load_balancer(&packet);
```

2. Discovering redundancies

- Analyze combined program
- Identify conditions that hold for all executions

→ Identify dead branches
→ Identify constants in packet
→ …

3. Optimization

- Replace instructions in downstream program
- Run conventional compiler passes

→ Replace dead branch by jump
→ Replace constant read by value
→ Run dead code elimination

Optimizing Program Chains

1. Program combination

```
if (firewall(&packet) != DROP)
    load_balancer(&packet);
```

2. Discovering redundancies

- Analyze combined program
- Identify conditions that hold for all executions

→ Identify dead branches
→ Identify constants in packet
→ …

3. Optimization

- Replace instructions in downstream program
- Run conventional compiler passes

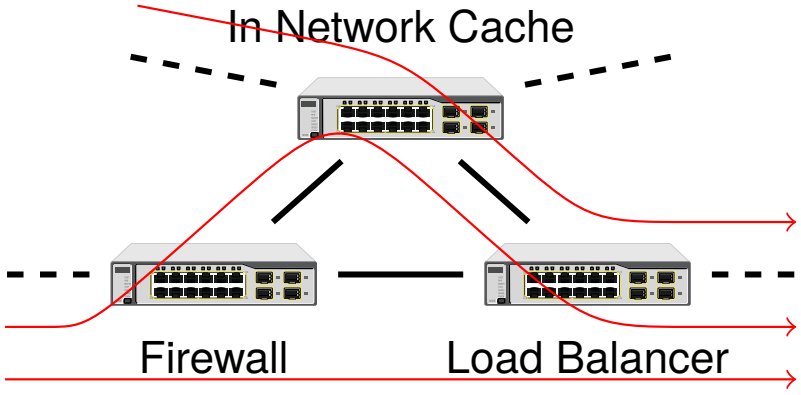
→ Replace dead branch by jump
→ Replace constant read by value
→ Run dead code elimination

Runtime to optimize the cilium.io Load Balancer in our example

- **30 s** for all steps including **25 s** of symbolic execution

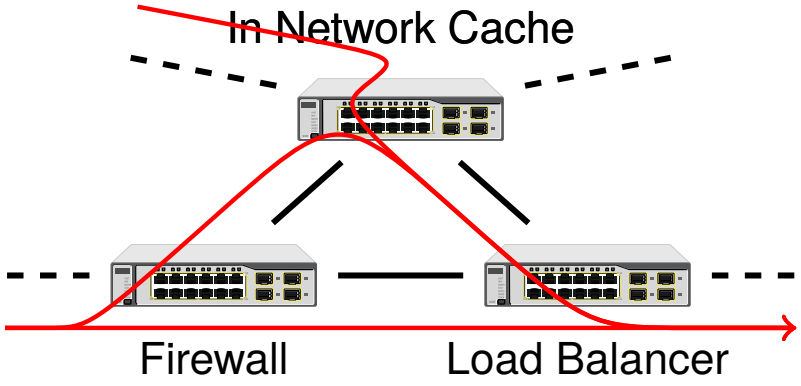
Challenge: Program Chain Selection

A program may be part of multiple network paths.



Challenge: Program Chain Selection

A program may be part of multiple network paths.

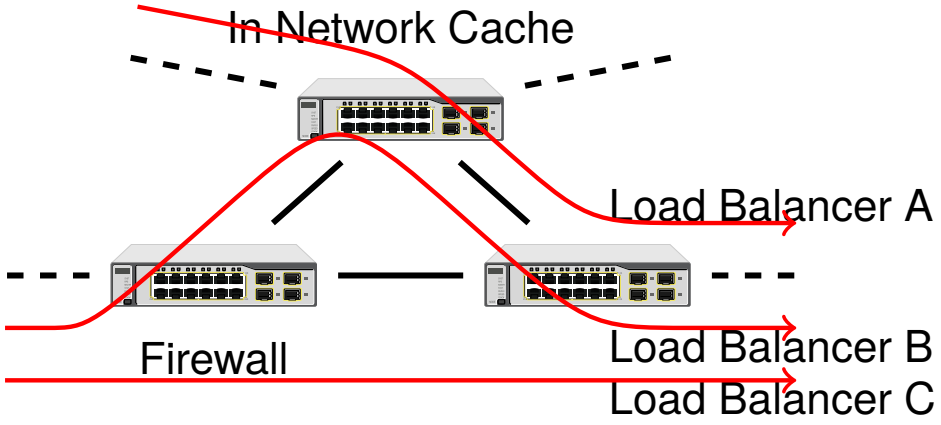


- **Optimizing for multiple paths**
  - A single optimized variant valid for all paths
  - Different optimized variants for different paths

Challenge: Program Chain Selection

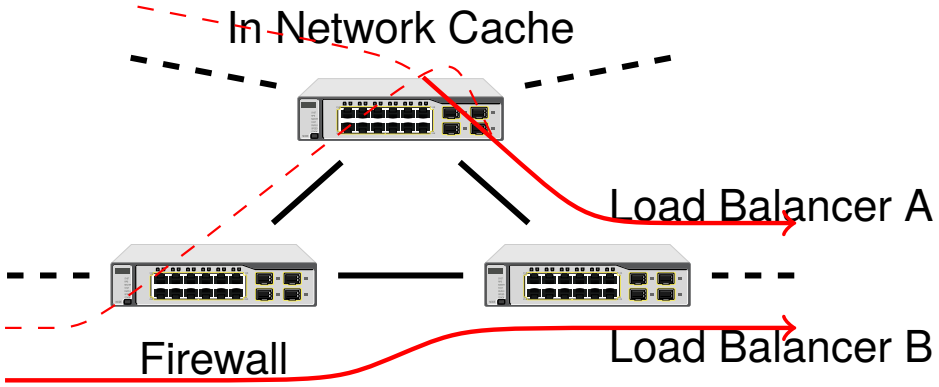
A program may be part of multiple network paths.

- **Optimizing for multiple paths**
  - A single optimized variant valid for all paths
  - Different optimized variants for different paths



Challenge: Program Chain Selection

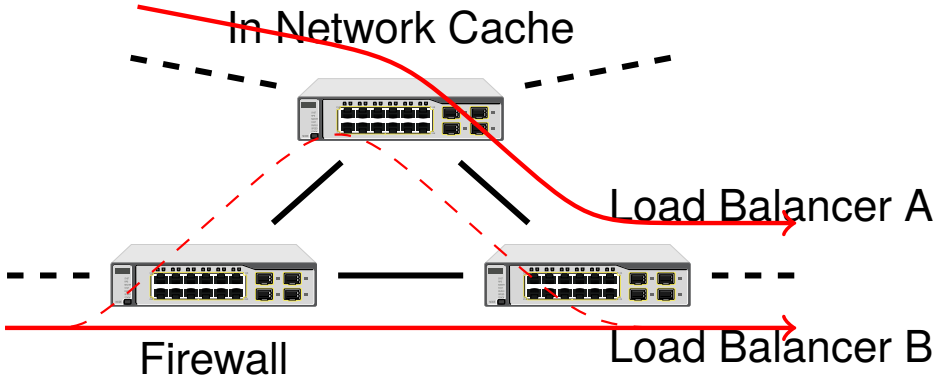
A program may be part of multiple network paths.



- **Optimizing for multiple paths**
  - A single optimized variant valid for all paths
  - Different optimized variants for different paths
- **Ignore some programs during optimization**
  - Only consider neighbours
  - Leave out irrelevant intermediate nodes

Challenge: Program Chain Selection

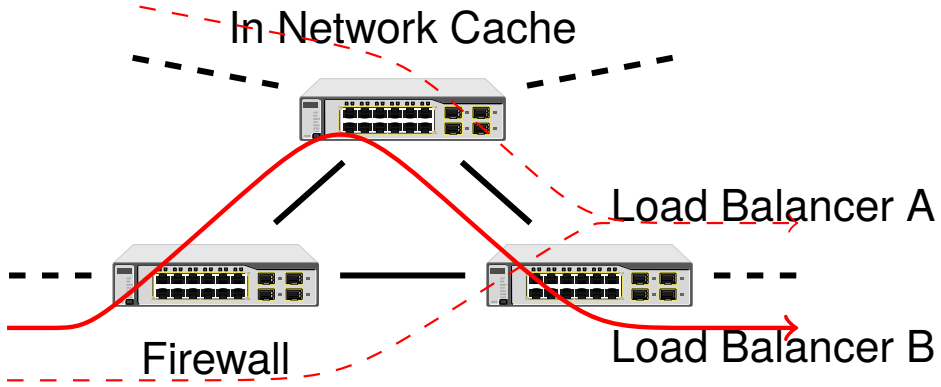
A program may be part of multiple network paths.



- **Optimizing for multiple paths**
  - A single optimized variant valid for all paths
  - Different optimized variants for different paths
- **Ignore some programs during optimization**
  - Only consider neighbours
  - Leave out irrelevant intermediate nodes

Challenge: Program Chain Selection

A program may be part of multiple network paths.



- **Optimizing for multiple paths**
  - A single optimized variant valid for all paths
  - Different optimized variants for different paths
- **Ignore some programs during optimization**
  - Only consider neighbours
  - Leave out irrelevant intermediate nodes
- **Selectively applying optimizations**
  - Optimize slow hotspots
  - Identify program chains with high optimization potential
  - Run some optimization whenever controller is idle

Summary

- **Our proposal: Integrating the network topology into program optimization**

- **High optimization potential on example with real-world program**
  - 2.3× packet rate increase
  - 88% program size reduction
- **Still much to do**
  - Improving optimization time
  - Considering multiple network paths
  - Evaluating on realistic networks

Summary

- **Our proposal: Integrating the network topology into program optimization**

- **High optimization potential on example with real-world program**
  - 2.3× packet rate increase
  - 88% program size reduction
- **Still much to do**
  - Improving optimization time
  - Considering multiple network paths
  - Evaluating on realistic networks

Questions?