Distributed Network Monitoring and Debugging with SwitchPointer

Praveen Tammana†  Rachit Agarwal‡  Myungjin Lee†

†University of Edinburgh, ‡Cornell University
Network monitoring and debugging is complex
Network monitoring and debugging is complex

Marple [SIGCOMM’17], PathDump [OSDI’16], FlowRadar [NSDI’16], EverFlow [SIGCOMM’15], Trumpet [SIGCOMM’16], Opensketch [NSDI’13]
Network monitoring and debugging is complex

Marple [SIGCOMM’17], PathDump [OSDI’16], FlowRadar [NSDI’16], EverFlow [SIGCOMM’15], Trumpet [SIGCOMM’16], Opensketch [NSDI’13]

• Increasingly larger scale
  • Over 100k endpoints
  • 10/40/100 GE
  • Aggregate traffic > 100 Tbps
An example: Too many red lights

F\(_1\), F\(_2\), F\(_3\): High priority
F\(_1\): Low priority
An example: Too many red lights

F₁: Low priority
F₂, F₃: High priority
An example: Too many red lights

F₁: Low priority
F₂, F₃: High priority

F₁ throughput

At S₁

At S₂

Gbps

msec

msec
An example: Too many red lights

F_1: Low priority
F_2, F_3: High priority

F_1 throughput

At S_1

At S_2

Gbps

msec

msec
An example: Too many red lights

F₁: Low priority
F₂, F₃: High priority

F₁ throughput

~ 600 Mbps

At S₁

At S₂

~ 600 Mbps
An example: Too many red lights

F₁: Low priority
F₂, F₃: High priority
An example: Too many red lights

F1: Low priority
F2, F3: High priority

F1 throughput

At S1

At S2

Gbps

0 0.2 0.4 0.6 0.8 1

0 2 4 6 msec

~ 200 Mbps
An example: Too many red lights

F1: Low priority
F2, F3: High priority

F1, F2 at S1
F1, F3 at S2
An example: Too many red lights

F₁: Low priority
F₂, F₃: High priority

At S₁
F₁, F₂ at S₁ and their packet priorities

At S₂
F₁, F₂ at S₁, F₁, F₃ at S₂
Existing Approaches

In-network techniques
Existing Approaches

In-network techniques
Existing Approaches

In-network techniques
Existing Approaches

In-network techniques

High in-network visibility

Requires more data plane resources

E.g.: Marple, EverFlow, FlowRadar
Existing Approaches

In-network techniques

- High in-network visibility
- Requires more data plane resources
- E.g.: Marple, EverFlow, FlowRadar

End-host based techniques

- More resources & programmability
- Lose network visibility
- E.g.: PathDump, Trumpet
In-network techniques

End-host based techniques

Requires more data plane resources
E.g.: Marple, EverFlow, FlowRadar

More resources & programmability
E.g.: PathDump, Trumpet

High in-network visibility

Lose network visibility

SwitchPointer
SwitchPointer
Integrates the best of two worlds

Insight: End-hosts collect and monitor telemetry data
SwitchPointer
Integrates the best of two worlds

Insight: End-hosts collect and monitor telemetry data

New insight: Switch stores the telemetry data
SwitchPointer
Integrates the best of two worlds

Insight: End-hosts collect and monitor telemetry data

New insight: Switch stores the telemetry data **pointers to end-hosts**
SwitchPointer
Integrates the best of two worlds

Insight: End-hosts collect and monitor telemetry data

New insight: Switch stores the telemetry data pointers to end-hosts
SwitchPointer
Integrates the best of two worlds

Insight: End-hosts collect and monitor telemetry data
New insight: Switch stores the telemetry data pointers to end-hosts
SwitchPointer in a nutshell
SwitchPointer in a nutshell

- Divides time into epochs
- Maintains per-epoch pointer to all end-hosts
- Embeds linkID(■) and epochID (■)
SwitchPointer in a nutshell

- Divides time into epochs
- Maintains per-epoch pointer to all end-hosts
- Embeds linkID() and epochID ( )

- Collect and monitor telemetry data
- Provides query service to filter telemetry data
SwitchPointer in a nutshell

- Divides time into epochs
- Maintains per-epoch pointer to all end-hosts
- Embeds linkID (■) and epochID (■)

- Collect and monitor telemetry data
- Provides query service to filter telemetry data

- Uses pointers at switches
- Locates the data necessary for debugging
Let’s revisit “Too many red lights” problem
Let’s revisit “Too many red lights” problem
Let’s revisit “Too many red lights” problem
Let’s revisit “Too many red lights” problem
Let’s revisit “Too many red lights” problem
Let’s revisit “Too many red lights” problem
Let’s revisit “Too many red lights” problem

F₁, F₂ contend at S₁
F₁, F₃ contest at S₂

S₁, S₂, S₃ EpochIDs
SwitchPointer: Four technical challenges

• How to decide the right epoch size?

• How to efficiently maintain pointers?

• How to efficiently embed telemetry data?

• How to handle asynchronous clocks?
SwitchPointer design

Data structure for pointers

Tradeoff between memory and bandwidth, and system efficiency
SwitchPointer design

Data structure for pointers

Tradeoff between memory and bandwidth, and system efficiency

Small epoch size
SwitchPointer design

Data structure for pointers

Tradeoff between memory and bandwidth, and system efficiency

Per-epoch pointers

Large memory

Efficient

Small epoch size

Data plane
SwitchPointer design

Data structure for pointers

Tradeoff between memory and bandwidth, and system efficiency

Control plane

Large bandwidth

Small epoch size

Efficient
SwitchPointer design

Data structure for pointers

Tradeoff between memory and bandwidth, and system efficiency

Large epoch size

Small bandwidth

Small memory

Inefficient
SwitchPointer design

Our solution: Hierarchical data structure for pointers
Each subsequent level has epochs with exponentially larger time scales
SwitchPointer design

Our solution: Hierarchical data structure for pointers
Each subsequent level has epochs with exponentially larger time scales

Epoch size = \( \alpha \)
SwitchPointer design

Our solution: Hierarchical data structure for pointers
Each subsequent level has epochs with exponentially larger time scales

Epoch size = $\alpha$

Level 1

$\alpha$ ms $\alpha$ ms ... $\alpha$ ms

$\alpha$ set of pointers
SwitchPointer design

Our solution: Hierarchical data structure for pointers
Each subsequent level has epochs with exponentially larger time scales

Epoch size = $\alpha$

Level 2

$\alpha^2$ ms

Level 1

$\alpha$ ms $\alpha$ ms ... $\alpha$ ms

$\alpha$ set of pointers

$\alpha$ set of pointers

$\alpha$ set of pointers
Our solution: Hierarchical data structure for pointers
Each subsequent level has epochs with exponentially larger time scales

SwitchPointer design

Epoch size = $\alpha$

- Level $k$: $\alpha^k$ ms
- Level 2: $\alpha^2$ ms
- Level 1: $\alpha$ ms, $\alpha$ ms, ..., $\alpha$ ms

$\alpha$ set of pointers
SwitchPointer design

Our solution: Hierarchical data structure for pointers

\[ \alpha = 10 \text{ ms} \quad k = 3 \]

Level 3

1000 ms

Level 2

100 ms .... 100 ms

Level 1

10 ms 10 ms ... 10 ms

10 set of pointers

10 set of pointers

10 set of pointers
SwitchPointer design

Our solution: Hierarchical data structure for pointers

\[ \alpha = 10 \text{ ms} \quad k = 3 \]

- Level 3: 1000 ms
- Level 2: 100 ms
- Level 1: 10 set of pointers

10 set of pointers
SwitchPointer design

Our solution: Hierarchical data structure for pointers

\[ \alpha = 10 \text{ ms} \quad k = 3 \]

Level 3

Redundant information

Level 2

100 ms

Level 1

10 ms 10 ms \ldots 10 ms

10 set of pointers
SwitchPointer design

Our solution: Hierarchical data structure for pointers

\[ \alpha = 10 \text{ ms} \quad k = 3 \]

Level 3

Redundant information

Level 2

100 ms

Level 1

10 ms 10 ms \ldots 10 ms

10 set of pointers

Storage \approx N \times \alpha \times K
SwitchPointer design

Our solution: Hierarchical data structure for pointers

\[ \alpha = 10 \text{ ms} \quad k = 3 \]

Level 3

✓ 100k end-hosts : 345KB

Level 2

100 ms

Level 1

10 ms 10 ms 10 ms

10 set of pointers

Storage \approx N \times \alpha \times K
SwitchPointer design

Our solution: Hierarchical data structure for pointers

\[ \alpha = 10 \text{ ms} \quad k = 3 \]

Level 3

Level 2

Level 1

10 set of pointers

Fine grained view
SwitchPointer design

Our solution: Hierarchical data structure for pointers

\[ \alpha = 10 \text{ ms} \quad k = 3 \]
SwitchPointer design

Our solution: Hierarchical data structure for pointers

$\alpha = 10\ \text{ms} \quad k = 3$

Control plane

Push top-level pointers

Level 3

1000 ms

Level 2

100 ms

Level 1

10 ms 10 ms ... 10 ms

10 set of pointers
SwitchPointer design

Our solution: Hierarchical data structure for pointers

\[ \alpha = 10 \text{ ms} \quad k = 3 \]

Control plane

✔️ 100k end-hosts: 100 Kbps

Level 3: 1000 ms

Level 2: 100 ms ...

Level 1: 10 ms 10 ms 10 ms

10 set of pointers
SwitchPointer: Four technical challenges

- How to decide the right epoch size?
- How to efficiently maintain pointers?
- How to efficiently embed telemetry data?
- How to handle asynchronous clocks?
Minimal Perfect Hash Functions

- Maps distinct keys (dest IPs) to a set of integers
- No hash collisions
- 2.1 bits of storage per end-host
- Construction time is large
SwitchPointer design

Maintaining pointers in the hierarchical data structure

Minimal perfect hash function (MPHF)

Level k

111011

Level 2

101001

Level 1

001000

MPHF

Pointer
SwitchPointer design

Maintaining pointers in the hierarchical data structure

Minimal perfect hash function (MPHF)

• Single operation to find the index to set in all levels
SwitchPointer design

Maintaining updated pointers in the hierarchical data structure

Minimal perfect hash function (MPHF)

dstIP\_1  dstIP\_2  \ldots  dstIP\_n

Lookup using MPHF

011001

Checks dstIP’s corresponding bit in the bit array
SwitchPointer: Four technical challenges

• How to decide the right epoch size?

• How to efficiently maintain pointers?

• How to efficiently embed telemetry data?

• How to handle asynchronous clocks?
SwitchPointer design

Switch embeds telemetry data (e.g., linkID, epochID)
SwitchPointer design

Switch embeds telemetry data (e.g., linkID, epochID)

- **INT**: Packet header space limitation
- **Cherrypick** [SOSR’15] for current deployments
SwitchPointer design

Switch embeds telemetry data (e.g., linkID, epochID)

- INT: Packet header space limitation
- **Cherrypick** [SOSR’15] for current deployments
SwitchPointer design

Switch embeds telemetry data (e.g., linkID, epochID)

- **INT**: Packet header space limitation
- **Cherrypick** [SOSR'15] for current deployments
SwitchPointer design

• Switch embeds telemetry data (e.g., linkID, epochID)
  • Packet header space limitation
  • Cherrypick [SOSR’15] for current deployments

• End-host collect and monitor telemetry data (E.g., PathDump [OSDI’16])
SwitchPointer design

• Switch embeds telemetry data (e.g., linkID, epochID)
  • Packet header space limitation
  • Cherrypick [SOSR’15] for current deployments

• End-host collect and monitor telemetry data (E.g., PathDump [OSDI’16])
  • Reconstructs the path
  • Computes a range of epochs for pod switches
SwitchPointer design

- Switch embeds telemetry data (e.g., linkID, epochID)
  - Packet header space limitation
  - **Cherrypick** [SOSR’15] for current deployments

- End-host collect and monitor telemetry data (E.g., PathDump [OSDI’16])
  - Reconstructs the path
  - Computes a range of epochs for pod switches

More details in our paper
SwitchPointer design

• Switch embeds telemetry data (e.g., linkID, epochID)
  • Packet header space limitation
  • Cherrypick [SOSR’15] for current deployments

• End-host collect and monitor telemetry data (E.g., PathDump [OSDI’16])
  • Reconstructions the path
  • Computes a range of epochs for pod switches

INT simplifies embedding and decoding telemetry data

More details in our paper
SwitchPointer: Four technical challenges

• How to decide the right epoch size?

• How to efficiently maintain pointers?

• How to efficiently embed telemetry data?

• How to handle asynchronous clocks?

  Set bound on clock difference between any pair of devices
SwitchPointer: Four technical challenges

• How to decide the right epoch size?

• How to efficiently maintain pointers?

• How to efficiently embed telemetry data?

• How to handle asynchronous clocks?

  Set bound on clock difference between any pair of devices

More details in our paper
SwitchPointer - Coverage
SwitchPointer - Coverage

In-network techniques

• TCP in-cast diagnosis
• Heavy hitter
• ECMP load imbalance diagnosis
• Silent random packet drops
• Traffic matrix
• DDoS

::
SwitchPointer - Coverage

**In-network techniques**
- TCP in-cast diagnosis
- Heavy hitter
- ECMP load imbalance diagnosis
- Silent random packet drops
- Traffic matrix
- DDoS
  
**End-host based techniques**
- TCP out of order packet delivery
- TCP non-monotonic
- Traffic bursts
- SYN flood attacks
- New TCP connections
- TCP in-complete flows
### SwitchPointer - Coverage

#### In-network techniques
- TCP in-cast diagnosis
- Heavy hitter
- ECMP load imbalance diagnosis
- Silent random packet drops
- Traffic matrix
- DDoS
  - ...

#### End-host based techniques
- TCP out of order packet delivery
- TCP non-monotonic
- Traffic bursts
- SYN flood attacks
- New TCP connections
- TCP in-complete flows
  - ...

### Spatially and temporally correlated problems
- E.g.: Too many red lights, Traffic cascades
### SwitchPointer - Coverage

<table>
<thead>
<tr>
<th>In-network techniques</th>
<th>End-host based techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>• TCP in-cast diagnosis</td>
<td>• TCP out of order packet delivery</td>
</tr>
<tr>
<td>• Heavy hitter</td>
<td>• TCP non-monotonic</td>
</tr>
<tr>
<td>• ECMP load imbalance diagnosis</td>
<td>• Traffic bursts</td>
</tr>
<tr>
<td>• Silent random packet drops</td>
<td>• SYN flood attacks</td>
</tr>
<tr>
<td>• Traffic matrix</td>
<td>• New TCP connections</td>
</tr>
<tr>
<td>• DDoS</td>
<td>• TCP in-complete flows</td>
</tr>
</tbody>
</table>

Spatially and temporally correlated problems
E.g.: Too many red lights, Traffic cascades

https://github.com/PathDump/Applications
Problems SwitchPointer cannot debug

- Instantaneous queue sizes
- Overlay loop detection
- Incorrect packet modification
- Packet properties at a switch

https://github.com/PathDump/Applications
A more complex example: Traffic cascades

F_1: High priority
F_2: Middle priority
F_3: Low priority
A more complex example: Traffic cascades

F1: High priority
F2: Middle priority
F3: Low priority
A more complex example: Traffic cascades

F₁: High priority
F₂: Middle priority
F₃: Low priority

Gbps

msec
A more complex example: Traffic cascades

F1: High priority
F2: Middle priority
F3: Low priority

Gbps

0 10 20 40
msec

F1

F2

F3
A more complex example: Traffic cascades

F₁: High priority
F₂: Middle priority
F₃: Low priority
A more complex example: Traffic cascades

F1: High priority
F2: Middle priority
F3: Low priority
A more complex example: Traffic cascades

F₁: High priority
F₂: Middle priority
F₃: Low priority
A more complex example: Traffic cascades

- **F₁**: High priority
- **F₂**: Middle priority
- **F₃**: Low priority
A more complex example: Traffic cascades

F1: High priority
F2: Middle priority
F3: Low priority

F1, F2 collision
A more complex example: Traffic cascades

F₁: High priority
F₂: Middle priority
F₃: Low priority
A more complex example: Traffic cascades

- **F₁**: High priority
- **F₂**: Middle priority
- **F₃**: Low priority
A more complex example: Traffic cascades

F₁: High priority
F₂: Middle priority
F₃: Low priority
A more complex example: Traffic cascades

F₁: High priority
F₂: Middle priority
F₃: Low priority

F₂, F₃ collision
A more complex example: Traffic cascades

F₁: High priority
F₂: Middle priority
F₃: Low priority
A more complex example: Traffic cascades

F₁: High priority
F₂: Middle priority
F₃: Low priority
A more complex example: Traffic cascades

\[ F_1: \text{High priority} \]
\[ F_2: \text{Middle priority} \]
\[ F_3: \text{Low priority} \]
A more complex example: Traffic cascades

F₁: High priority
F₂: Middle priority
F₃: Low priority
A more complex example: Traffic cascades

F₁: High priority
F₂: Middle priority
F₃: Low priority

Query / Response
A more complex example: Traffic cascades

- **F1**: High priority
- **F2**: Middle priority
- **F3**: Low priority

- **F2** contends with **F3**
- **F1** contends with **F2**
A more complex example: Traffic cascades

**F1**: High priority
**F2**: Middle priority
**F3**: Low priority

F2 contends with F3
F1 contends with F2
A more complex example: Traffic cascades

F1: High priority
F2: Middle priority
F3: Low priority
SwitchPointer overhead (software implementation)

- Prototype
  - ✓ Implemented on top of OVS-DPDK version
  - ✓ Build minimal perfect hash function using CMPH library

![Graph showing throughput vs packet size]

- Throughput (Gbps)
- Packet size (Bytes)
- OVS
- SwitchPointer (#levels = 1)
- SwitchPointer (#levels = 5)
SwitchPointer overhead (software implementation)

- Prototype
  - Implemented on top of OVS-DPDK version
  - Build minimal perfect hash function using CMPH library

![Graph showing throughput loss for average packet size ≥ 256 Bytes]

- No throughput loss for average packet size ≥ 256 Bytes
Conclusion

• Achieves benefits of both end-host and in-network approaches

• Switch acts as a “directory service”

• Uses end-host resources to collect and monitor telemetry data

• Debugs a large class problems

• Ongoing work: Hardware implementation using P4 and NetFPGA