SRv6Pipes: enabling in-network bytestream functions

Fabien Duchêne <fabien.duchene@uclouvain.be>
David Lebrun <david.lebrun@uclouvain.be>
Olivier Bonaventure <olivier.bonaventure@uclouvain.be>

IFIP Networking 2018 - Zurich
SRv6Pipes?

SRv6: IPv6 Segment Routing

Pipes: Unix pipeline
IPv6 Segment Routing

- IPv6 flavor of a modern variant of the source routing paradigm
- Extension header, named Segment Routing Header (SRH)
- Each segment is an IPv6 address representing a node or link to traverse
IPv6 Segment Routing

2001:1234::AAAA

2001:1234::BBBB

2001:1234::CCCC
IPv6 Segment Routing

2001:1234::AAAA

2001:1234::BBBB

2001:1234::CCCC
IPv6 Segment Routing

2001:1234::AAAA

2001:1234::BBBB

2001:1234::CCCC

IGP Path
SRv6 Path
### IPv6 Segment Routing Header (SRH)

The IPv6 Segment Routing Header (SRH) is a header used in IPv6 to segment routing. It allows for more flexible routing options than the traditional IPv6 route header.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next Header</td>
<td>Header Information</td>
<td></td>
</tr>
<tr>
<td>Hdr Ext Len</td>
<td>Header Extension Length</td>
<td></td>
</tr>
<tr>
<td>Routing Type</td>
<td>Routing Type (0 = local, 1 = global)</td>
<td></td>
</tr>
<tr>
<td>Segments Left</td>
<td>Number of remaining segments</td>
<td></td>
</tr>
<tr>
<td>Last Entry</td>
<td>Last Entry Information</td>
<td></td>
</tr>
<tr>
<td>Flags</td>
<td>Flags for routing</td>
<td></td>
</tr>
<tr>
<td>Tag</td>
<td>Tag for routing</td>
<td></td>
</tr>
</tbody>
</table>

**Segment List:**

- Segment List[0] (128 bits IPv6 address)
  - `2001:1234::CCCC`
- Segment List[n] (128 bits IPv6 address)
  - `2001:1234::BBBB`
- Segment List[n] (128 bits IPv6 address)
  - `2001:1234::AAAA`

**Optional Type Length Value objects (variable):**
IPv6 Segment Routing Header (SRH)

Segment List[0] (128 bits IPv6 address)
2001:1234::CCCC

Segment List[n] (128 bits IPv6 address)
2001:1234::BBBB

Segment List[n] (128 bits IPv6 address)
2001:1234::AAAA

Optional Type Length Value objects (variable)
IPv6 Segment Routing Header (SRH)

Segment List[0] (128 bits IPv6 address)
2001:1234::CCCC

... 2001:1234::BBBB

Segment List[n] (128 bits IPv6 address)
2001:1234::AAAA

Optional Type Length Value objects (variable)
IPv6 Segment Routing

2001:1234::AAAA

::CCCC
::BBBB
::AAAA

2001:1234::BBBB

2001:1234::CCCC
IPv6 Segment Routing

2001:1234::AAAA

2001:1234::BBBB

::CCCC
::BBBB
::AAAA

2001:1234::CCCC
Unix pipeline

```bash
fab@x240-fab:~$ echo "I love dogs!"
I love dogs!
fab@x240-fab:~$
```
Unix pipeline

```
fab@x240-fab:-$ echo "I love dogs!"
I love dogs!
fab@x240-fab:-$ echo "I love dogs!" | base64
S5Bwb3ZlIQRvZ3MhCg==
fab@x240-fab:-$
```
Unix pipeline

```bash
fab@x240-fab:~$ echo "I love dogs!"
I love dogs!
fab@x240-fab:~$ echo "I love dogs!" | base64
SSBsb3ZlIGRvZ3MhCg==
fab@x240-fab:~$ echo "I love dogs!" | base64 | sed 's/RvZ3/NhdH/g'
SSBsb3ZlIGNhdHMhCg==
fab@x240-fab:~$
```
Unix pipeline

```
fab@x240-fab:~$ echo "I love dogs!"
I love dogs!
fab@x240-fab:~$ echo "I love dogs!" | base64
SSBsb3ZlIGRvZ3NhCg==
fab@x240-fab:~$ echo "I love dogs!" | base64 | sed 's/RvZ3/NhdH/g'
SSBsb3ZlIGNhdHkHMHc==
fab@x240-fab:~$ echo "I love dogs!" | base64 | sed 's/RvZ3/NhdH/g' | base64 -d
I love cats!
fab@x240-fab:~$
```
Unix **pipeline**

```
# Unix pipeline example

fab@x240-fab:-->$ echo "I love dogs!"
I love dogs!
fab@x240-fab:-->$ echo "I love dogs!" | base64
SSBsb3ZlIGRvZ3MhCg==
fab@x240-fab:-->$ echo "I love dogs!" | base64 | sed 's/RvZ3/NhdH/g'
SSBsb3ZlIGNhdmFhCg==
fab@x240-fab:-->$ echo "I love dogs!" | base64 | sed 's/RvZ3/NhdH/g' | base64 -d
I love cats!
fab@x240-fab:-->$
```
Leverage IPv6 Segment Routing to allow the user to **choose and apply a chain of functions** to the **payload**.
SRv6 + pipes = SRv6Pipes!

My secret message

Encrypted

This is a pipe :)
Enabling in-network *bytestream* functions

Middleboxes can perform two different types of network functions:

- **per-packet** functions: operate on a per-packet basis
  - Network Address Translation (NAT), stateless firewall….
  - operate on the network and sometimes transport header
  - simple

- **per-bytestream** functions: operate on the payload of the TCP packets
  - compression, encryption, transcoding…
  - **reorder** the received TCP packets and often modify the payload of TCP packets
  - include an almost complete **TCP implementation**
  - complex
Transparent TCP Proxy

TCP connections
- Green: Client -> Proxy
- Red: Proxy -> Server

Client

Proxy

Buffer

Dogs

Cats

send()

receive()

Dogs

Modify

Cats
SRv6Pipes: almost ready!

Encrypt

Proxy

Decrypt

2001:1234::CCCC

2001:1234::CCCC
SRv6Pipes : modular transformation

The client should be able to create any chain of functions.

How to represent a function?

- One proxy per function is too expensive
- One proxy should be able to perform several functions

The client must be able to inform the proxy about the function
We leverage the large addressing space available in IPv6.

- Each **proxy announces** one or more **IPv6 prefixes**
- Allocate a given amount of bits to encode the **identifier of the function**
- The remaining **low order bits** are used to specify **parameters** of the function

\[
\]

- **2001:0123:4567:8901:2345/80** : Proxy range
- **AAAA** : Identifier of the function
- **BCDE:FFFF** : Parameters of the function

Encoding Functions and Parameters

Encrypt

Decrypt

Transcode

Modified Payload

AAAA

BBBB

CCCC

Lookup function ID

Unmodified Payload

Proxy

SRv6Pipes: the big picture

Encrypt


Proxy

Decrypt


Proxy

Encrypt

EndHost

Decrypt

Encrypt

2001:1234::CCCC
SRv6Pipes: other design points

How is the return traffic handled?

We insert a Type Length Value (TLV) object containing the return path in the original SRH.

How does the client get informations about the addresses of the proxies?

We modify the DNS resolver. This is detailed in [1].

SRv6Pipes : implementation

Implementation details :

- Modification of the Linux kernel version 4.16.0
- Modification of iproute2
- Implementation of the proxy (~1000 C lines)
  - Uses NFQUEUE to intercept the SYN and extract the SRH
  - Leverages TPROXY to establish and transparent connection
  - Uses ip6tables
  - Allows to load dynamic modules to support new functions

Runs on commodity hardware.

Implementation: performance evaluation

Client: 2.53Ghz Intel Xeon 16GB RAM - Debian Stretch - Kernel 4.16 - wrk 4.0.2-5
Middleboxes/Server: 2.53Ghz Intel Xeon 8GB RAM - Debian Stretch - Kernel 4.16 - lighttpd 1.4.35
Measurements Results: no loss/delay

- 1 middlebox acting as proxy/router
- 200 clients downloading web pages of a given size.

- For 10MB files:
  - Proxy: ~9840Mb/s
  - Router: ~9840Mb/s

- For 1KB files:
  - Proxy: 253Mb/s
  - Router: 272Mb/s

Cost of establishment.
Measurements Results: with loss and delay

- Same setup
- Loss: 1% per link
- Delay 1% per link

- Our proxy acts as a Performance Enhancing Proxy (PEP)
Measurements Results: no loss/delay - 2 proxies

- 2 proxies/routers
- Applying a XOR function
- No significant overhead
Conclusion

SRv6 pipes:

- Middleboxes are explicitly exposed
- **Flexibility** for the Network Operators
- In-network **per-bytestream** and **per-packet** functions
- **New use cases** for IPv6 Segment Routing
- **Implemented** in the Linux kernel and available today!
Thank you!

Fabien Duchêne <fabien.duchene@uclouvain.be>
Measurements Results: CPU intensive functions

![Graph showing transfer rates with different XOR executions. The graph plots rate (Mbits/sec) against page size (1KB to 10MB) for 1, 2, 4, 8, and 16 executions.}
Implementation details

(a) Traversal of a SYN packet through the proxy. The SRH is recorded for the 5-tuple.

(b) Traversal of data packets.

(c) Traversal of return packets.