DE·CIX

jFlowLib

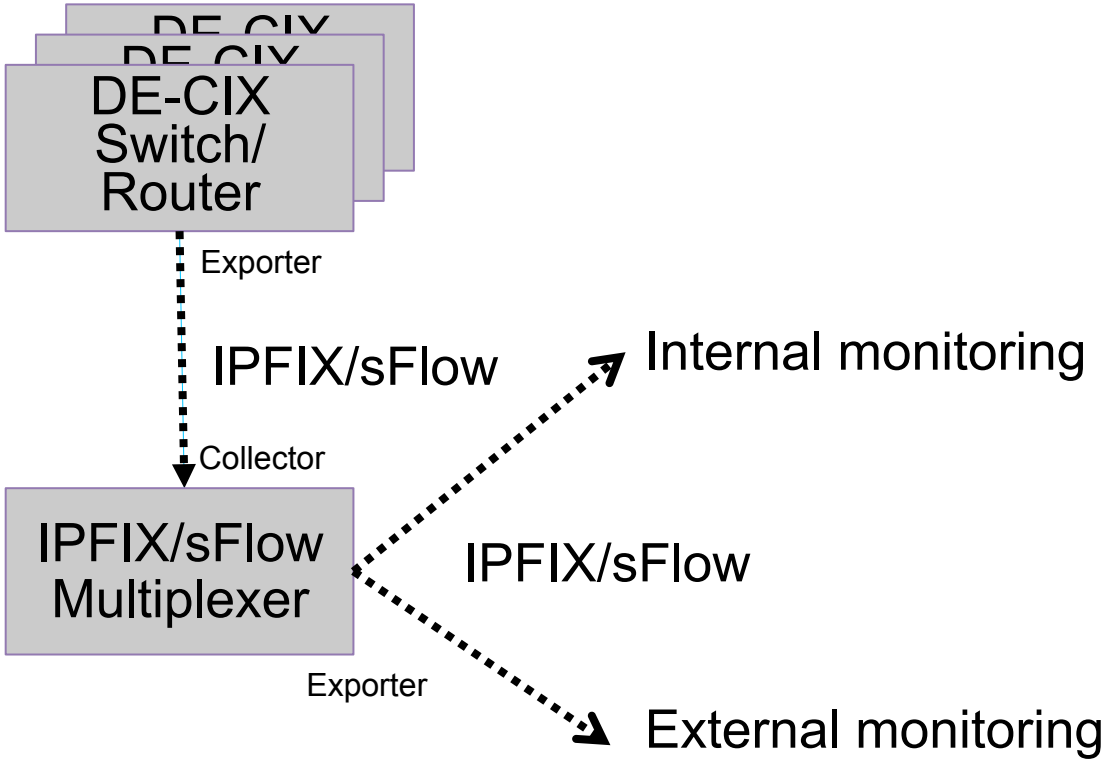
A Java Library to Parse and Generate sFlow and IPFIX Data

Dr. Thomas King
Manager R&D

jFLowLib

- jsFlow:
  - Java library for sFlow (v5): Counter and Sampling
  - Supports sFlow from Force10 E-series and Alcatel Lucent 7750 (at least)
- jIPFIX:
  - Java library for IPFIX: L2IP Template
  - Supports IPFIX Alcatel Lucent 7750 (at least)

Motivation


DE-CIX
Switch/
Router
Exporter
IPFIX/sFlow
Collector
IPFIX/sFlow
Multiplexer
Exporter
Internal monitoring
IPFIX/sFlow
External monitoring

Main Use Case

Multiplexing IPFIX/sFlow:

- 5000+ pkt/s IPFIX
- 1000+ pkt/s sFlow
- Without packet loss
- Many exporters (routers/switches) (up to 10)
- Many (changing) collectors (up to 10)
- IP spoofing (using RAWSockets)
- Configuration

Main Use Case

Snippet:

```
<?xml version="1.0"?>
<jflowlib-muxer>
<jipfix-muxer>
  <listen>
    <ip>10.102.200.5</ip>
    <port>2055</port>
    <startmissingdatarecorddetector>false</startmissingdatarecorddetector>
  </listen>
  <ping>
    <collectors>true</collectors>
    <ip>10.102.0.19</ip>
  </ping>
<!-- DX sflow-counter-dev -->
  <muxer type="plain">
    <collector>
      <ip>192.168.63.19</ip>
      <port>2056</port>
    </collector>
  </muxer>
</jipfix-muxer>
</jflowlib-muxer>

java -XX:+UseConcMarkSweepGC -Xmx1024m -Djava.library.path=/opt/jFlowLib/lib -jar jFlowLib.jar
-cfg /opt/jFlowLib/etc
```

Use Cases

Reading data from network:

Snippet:

```
DatagramSocket ds = new DatagramSocket(2055);
while (true) {
      byte[] data = new byte[65536];
      DatagramPacket dp = new DatagramPacket(data, data.length);
      ds.receive(dp);
      MessageHeader mh = MessageHeader.parse(dp.getData());
      System.out.println(mh);
}
```

Reading data from pcap files (using pcap4j):

Snippet:

```
PcapHandle handleRead = Pcaps.openOffline("test.pcap");
PacketListener listener = new PacketListener() {
      public void gotPacket(Packet fullPacket) {
            UdpPacket udpPacket = fullPacket.get(UdpPacket.class);
            byte[] onlyIPFIX = new byte[udpPacket.getRawData().length - 8];
            System.arraycopy(udpPacket.getRawData(), 8, onlyIPFIX, 0, bytes.length - 8);
            MessageHeader mh = MessageHeader.parse(onlyIPFIX);
            …
```

Use Cases II

Writing data to network:

Snippet:

```
DatagramSocket ds = new DatagramSocket(2055);
MessageHeader mh = new MessageHeader();
mh.setVersionNumber(10);
mh.setObservationDomainID(67108864);
mh.setSequenceNumber(seqNumber);
…
mh.setExportTime(new Date());
DatagramPacket dp = new DatagramPacket(mh.getBytes(), mh.getBytes().length,
collectorIPv4Value, collectorPortValue);
datagramSocket.send(dp);
```

Use Cases III

Writing data to pcap file (using pcap4j):

Snippet:

```
PcapDumper dumper = handleRead.dumpOpen("test.pcap");
MessageHeader mh = new MessageHeader();
mh.setVersionNumber(10);
mh.setObservationDomainID(67108864);
mh.setSequenceNumber(seqNumber);
…
UnknownPacket.Builder upB = new UnknownPacket.Builder();
upB.rawData(mh.getBytes());
udpB.payloadBuilder(upB);
Packet.Builder ipB = fullPacket.get(IpV4Packet.class).getBuilder();
ipB.payloadBuilder(udpB);
Packet.Builder etherB = fullPacket.get(EthernetPacket.class).getBuilder();
etherB.payloadBuilder(ipB);
Packet newPacket = etherB.build();
dumper.dump(newPacket, 0l, 0);
```

Use Cases IV

Anonymising IP addresses:

Random (IP_A -> IP_{Random}):

Snippet:

```
IPv4AddressRandomizer ipV4randomizer = new IPv4AddressRandomizer();
IPv6AddressRandomizer ipV6randomizer = new IPv6AddressRandomizer();
Inet4Address fakeDestIpv4 = (Inet4Address) ipV4randomizer.randomize(realDestIpv4);
Inet6Address fakeDestIpv6 = (Inet6Address) ipV6randomizer.randomize(realDestIpv6);
```

Pseudo-Random (IP_A -> IP_B):

Snippet:

```
IPv4AddressRandomizer ipV4randomizer = new IPv4AddressRandomizer(true);
IPv6AddressRandomizer ipV6randomizer = new IPv6AddressRandomizer(true);
Inet4Address fakeDestIpv4 = (Inet4Address) ipV4randomizer.randomize(realDestIpv4);
Inet6Address fakeDestIpv6 = (Inet6Address) ipV6randomizer.randomize(realDestIpv6);
```

Status

- License: Apache 2.0
- Source code: https://github.com/de-cix/jFlowLib
- 47 commits, 4 contributors
- Major parts are covered by software tests

- All relevant use cases for DE-CIX are implemented
- jFlowLib is actively used by DE-CIX
- DE-CIX will maintains jFlowLib in the future

- Your contribution is highly appreciated

Questions, Comments, Feedback?

DE·CIX

By joining DE-CIX, you become part of a universe of networks. Everywhere.

DE-CIX. Where networks meet.

DE-CIX Management GmbH
Lindleystr. 12
60314 Frankfurt
Germany
Phone +49 69 1730 902 0

sales@de-cix.net

www.de-cix.net

Thank you!