IPFIX in SerIoT: producing, collecting, and querying IP traffic statistics

Paweł Foremski, <pjf@iitis.pl>, IITiS PAN Gliwice
Version 1 (Feb 2019)

Abstract. This document discusses IP traffic statistics in SerIoT. The described system is based on the IPFIX standard (RFC7011, [1]), Open vSwitch (OVS, a reference OpenFlow switch software [2]), and InfluxDB (a time-series database [3]). We describe how OVS produces IP flow statistics, how we collect these statistics using IPFIX, and how we store them in InfluxDB. We also discuss how the collected data can be correlated with other subsystems of the SerIoT network, including the ONOS controller. The reader should learn how IPFIX works, how to collect IPFIX data from OVS, how to add new IPFIX traffic statistics, and how to query collected data for complex analyses, e.g. Anomaly Detection.

Table of contents

1. Introduction

SerIoT network requires a subsystem that collects and stores IP traffic statistics to provide network visibility (e.g. traffic visualization) and security (e.g. detecting anomalous traffic). In order to satisfy this requirement, we adopt the IPFIX standard defined in RFC 7011 [1], which is supported natively by OVS, the OpenFlow switch selected for the SerIoT project [2]. In order to collect the data produced by OVS, we implemented an open source IPFIX collector *ipflux* [22], which stores received data in InfluxDB [3].

The system is flexible and powerful, as it can be easily extended by adding a new IPFIX data producer that complies with the IANA assignments for IPFIX entities [4], and the system will immediately understand and store the new data. The powerful InfluxDB query language [5] allows for retrieving select information on IP flows, both historical and near real-time (delay of ~10 seconds), with multitudes of possible data aggregations. An overview of the system is given in the Fig. 1.1. below.

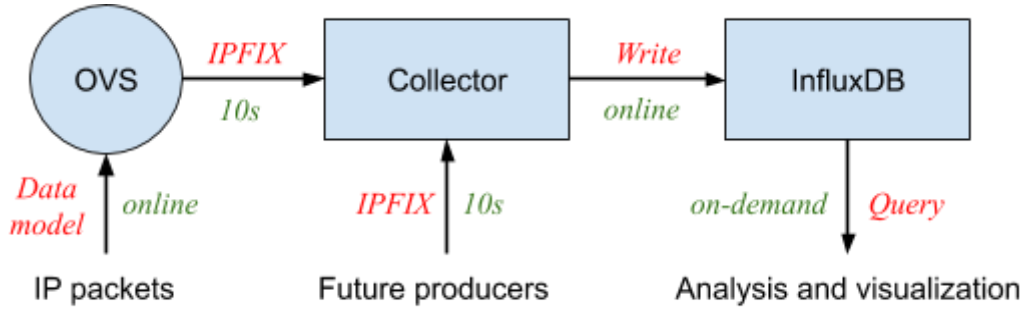

Fig. 1.1. Overview of the SerIoT IPFIX traffic data collection system. Interfaces and interactions shown in red, time delays in green.

Neither the system nor this document tackles the question on how to *make use* of the data. We merely deliver a generic tool that can be used for many tasks, and optionally extended to better suit some use cases. We explicitly set the topic of analyzing IP traffic data as out of scope of the IPFIX system and its documentation. However, note that the company behind InfluxDB provides a software stack for time-series data analysis, TICK [6], which may be useful in some applications. For instance, the Chronograf web interface [10] proved to be convenient for prototyping InfluxDB queries and monitoring live IPFIX data.

2. The IPFIX standard

The IPFIX standard is defined RFC 7011, and in fact it is just the next version of the Cisco NetFlow v9, i.e., IPFIX is the official name for NetFlow v10 that highlights the change from a proprietary protocol to an open, independent IETF standard [1,7,8].

It is crucial to understand that in essence IPFIX *is merely a data format*, like JSON or XML. It also specifies how to send and receive the IPFIX-encoded data over the network, but it does not produce any IP statistics on its own. Thus, one should understand that saying "the source of our data is IPFIX" is not a strictly valid statement. In fact, IPFIX is rather a popular mechanism of transporting IP statistics from *exporters* (or *producers*) to *collectors*.

That said, an accompanying standard *IPFIX data model* (specified in RFC 7012 [9]) discusses IP traffic statistics in the context of IPFIX. Foremost, it references the IANA standard "IP Flow IPFIX Entities" [4], a reference manual for working with IPFIX data. The document specifies common identifiers, names, and descriptions for ~500 possible characteristics of IP flows. For the SerIoT project, we adopt a subset of this IANA standard.

For a deep understanding of IPFIX we refer the reader to the RFC documents, but let us briefly describe the process of exporting and collecting IPFIX data in the context of SerIoT. Foremost, the basic source of IPFIX data is Open vSwitch (OVS), as discussed in section 3. The resultant IPFIX data is transported to the collector over the unreliable UDP protocol, which means the OVS will periodically re-transmit *IPFIX templates* to the collector every 10 minutes. The templates are a crucial piece of information that is required to decode IPFIX data. If the collector does not have the templates of a particular OVS switch, it will drop incoming data until it successfully receives the templates.

If the collector is started *after* OVS, it will miss the first transmission of IPFIX templates. Thus, in such a situation, expect up to 10 minute delay before the collector will be able to consume the incoming IPFIX data stream. However, if the collector is started *before* OVS, it should receive the first transmission of IPFIX templates, and the incoming IPFIX data should be stored in the database immediately after reception.

Also, keep in mind that IPFIX producers aggregate traffic statistics in time windows, e.g. 10 seconds, and send the data to collectors when a new piece of such time-aggregated data is ready. Thus, the information stored in the database for a particular flow will be updated every 10 seconds. However, this parameter may be changed in the configuration, but consider the possible increase in resource usage due to more data points.

3. Producing IPFIX data

We rely on an external open source project, Open vSwitch, to compute and provide us with the *basic* IP traffic statistics. If one needs more data, a new IPFIX producer must be implemented by the system user, according to the IANA IPFIX assignments [4].

3.1. Open vSwitch

Open vSwitch (OVS) is a "production quality, multilayer open virtual switch" [2] that supports the OpenFlow protocol [13]. It offers visibility into traffic via a plethora of mechanisms, including the IPFIX standard (explained in section 2). OVS implements a subset of the possible IPFIX entities described by IANA [4]. A copy of the IANA assignments can be found in the OVS source code under [12]. The IPFIX implementation can be found in the source code under ovs/ofproto-dpif-ipfix.c [11]. In particular, the function *ipfix_template_fields()* (line 1393 of that file) is responsible for generating an IPFIX template matching the IANA assignments, i.e. a subset of IP traffic features being tracked by the OVS. See appendix A of this document for a list of IP traffic features exported by OVS.

In order to enable and configure the IPFIX exporter in OVS, you must read the OVS documentation [16]. Understanding the OVS syntax and meaning is crucial to use the SerIoT IPFIX system properly. At minimum, please read the section "IPFIX TABLE" of ovs-vswitchd.conf.db(5) manual page [17] to understand the part of OVS configuration that needs an update, and the whole ovs-vsctl(8) manual page [18] to understand the tool that will be used to make that change. Note that contrary to many Linux system tools, OVS stores configuration changes permanently, so once done, they should work after a system reboot.

We explicitly set the topic of OVS configuration as out of scope of this document and the SerIoT IPFIX system. However, below please find an example of OVS configuration assuming you want to create a bridge "br0" that collects information on all packets that pass through:

```
# create bridge and add ports
ovs-vsctl add-br br0
ovs-vsctl add-port br0 eth1 -- set interface eth1 ofport=1
ovs-vsctl add-port br0 eth2 -- set interface eth2 ofport=2

# must be run on every system startup
ip link set dev eth1 up
ip link set dev eth2 up
ip link set dev br0 up

# enable IPFIX
ovs-vsctl -- set Bridge br0 ipfix=@i -- --id=@i create IPFIX \
  targets=\"192.168.100.1:4739\" obs_domain_id=Y obs_point_id=X \
  cache_active_timeout=10 cache_max_flows=100000 sampling=1
```

Listing 3.1. Illustrative IPFIX configuration for OVS

In the above example shown in Listing 3.1., we first create an OVS bridge “br0” out of interfaces “eth1” and “eth2”. Note we explicitly assign the “ofport” OVS property that matches the Linux interface suffix. We also run the iproute2 command “ip” to bring the interfaces up (which must be run on every system startup, e.g. from the rc.local script).

Then, we create a new entry in the OVS database table “IPFIX”, and link it to the “br0” entry of the “Bridge” table. This means, we add IPFIX configuration to the “br0” bridge. In the example, we set our IPFIX data collector address as 192.168.100.1, UDP port 4739 (make sure it is enabled in the firewall on the collector machine). As the IPFIX observation domain we use “Y”, and as the observation point we use “X”. Please set both Y and X to numeric identifiers, see Appendix C for recommendations.

Consult the “IPFIX TABLE” section of ovs-vswitchd.conf.db man page [17] to understand the meaning of the rest of the command line. Basically, we collect all packets without sampling, and aggregate 10 seconds of IP flow data until we send an IPFIX packet to the collector.

3.2. Adding new traffic statistics

If one needs more data than provided by OVS (listed in Appendix A), then instead of modifying OVS source code, we recommend writing or adapting an IPFIX exporter. Encoding data as IPFIX is relatively easy, and many software libraries exist for that purpose, e.g. for Python [19]. Note that although many powerful off-the-shelf IPFIX exporters exist, like nProbe [20] or pmacct [21], they may be an overkill for research purposes.

Thus, we recommend writing a small program that will collect just the information needed, encode as IPFIX using IANA assignments [4], and send the data to the ipflux collector, as presented in Fig. 1.1. The collector should automatically recognize the identifiers used in the templates, and store the data in the database, indexing as explained in Appendix B. One should consider using another IPFIX domain id, in order to distinguish the data coming from the new program from the data coming from the OVS.

4. Collecting IPFIX data

In order to collect IPFIX data, you need to install *ipflux* [22]. First, install the Golang compiler by visiting golang.org/dl. Under Debian-based distributions, you may just type "sudo apt-get install golang".

If you have Golang already installed, download ipflux and build the binary, e.g.:

```
# download
wget https://github.com/iitis/ipflux/archive/master.zip

# unpack
unzip master.zip

# build
cd ipflux-master
go build
```

Listing 4.1. Installing ipflux.

The last command should produce a command-line utility "ipflux-master" in the current directory. You may start it directly from your terminal screen, or under GNU Screen [23], for long-period data collection. By default, ipflux will connect to the database running on the local machine, and store data using simplified indexing. We will discuss InfluxDB installation and required command-line options later.

You may run "./ipflux-master -h" to get all possible options:

```
Usage of ./ipflux-master:
  -dbg int
       debugging level (default 1)
  -dropOut
       drop data on "output" flow direction (default true)
  -influx string
       InfluxDB server address (default "http://localhost:8086")
  -influxBatch int
       max InfluxDB write batch size (default 10000)
  -influxDB string
       InfluxDB database name (default "ipflux")
  -influxInsecure
       skip HTTPS cert validation for InfluxDB
  -influxPass string
       InfluxDB password
  -influxRP string
       InfluxDB retention policy
  -influxTime duration
       InfluxDB write interval (default 1s)
  -influxTimeout duration
```

```
      InfluxDB write timeout (default 15s)
  -influxUser string
      InfluxDB username
  -ipfixCache int
      IPFIX session cache size (default 1000)
  -keyInterface
      use interface name as InfluxDB tag
  -keyMAC
      use MAC addresses as InfluxDB tags
  -rcv int
      Receive packet buffer size (default 1000)
  -udp string
      UDP address to listen on (default ":4739")
```

Listing 4.2. ipflux command-line options.

The options starting with "-influx*" can be used to modify database connection. Note that by default all data will be stored in the database named "ipflux". Use the "-dbg" option to diagnose any problems. By default, "-dbg" is set to 1, which means ipflux will report on the standard error the number of data points written to the database (in real-time).

For the purpose of the SerIoT project, ipflux *must* be run with at least the below options, which respectively make it store both directions of traffic (ingress/egress) and use interface names and MAC addresses as InfluxDB tags.

```
./ipflux -dropOut=false -keyInterface -keyMAC
```

Listing 4.3. ipflux command-line for the SerIoT project.

5. Storing & querying IPFIX data

In order to store data in the database, you must have InfluxDB running. In order to do that, please follow the documentation at [24]. Also, please read introductory InfluxDB documentation at [27] to understand key InfluxDB concepts.

Before writing any data, we recommend switching the InfluxDB indexing engine from in-memory "TSM" to on-disk "TSI", as explained in [25]. Edit the InfluxDB configuration file "/etc/influxdb/influxdb.conf" and in the "[data]" section, set "index-version" to "tsi1".

Also, create a data retention policy that will periodically delete old data. If you skip this step, your disk and memory usage will grow with time. For example, do the following:

```
influx
CREATE DATABASE ipflux
CREATE RETENTION POLICY day ON ipflux DURATION 1d REPLICATION 1 DEFAULT
```

Listing 5.1. Creating InfluxDB data retention policy

Running the above will cause IPFIX data to be automatically deleted after 1 day. If you need to store the data for longer periods of time, please consult InfluxDB documentation on downsampling the data, e.g. [26].

Finally, let us store & query IPFIX data. First, ensure InfluxDB is running and you can connect to it, e.g. using the "influx" command-line tool. Then, ensure the "ipflux" collector is running and that UDP/4739 port is open in your firewall. Next, re-start your OVS switch to make sure the IPFIX templates will be sent to the collector. Finally, send some traffic through the bridge, e.g. ping another machine through the bridge. Wait 10 seconds, and you should be able to see similar results in your database:

```
$ influx
Connected to http://localhost:8086 version 1.7.3
InfluxDB shell version: 1.7.3
Enter an InfluxQL query
> USE ipflux
Using database ipflux
> SHOW MEASUREMENTS
name: measurements
name
----
domain1
> SHOW SERIES ON domain1
domain123,dir=in,dst_ip=192.168.17.1,dst_mac=10:eb:f8:79:10:02,ifname=eth1,obs_id=2,prot
o=1,src_ip=192.168.17.2,src_mac=10:42:17:c1:ce:02
domain123,dir=out,dst_ip=192.168.17.1,dst_mac=10:eb:f8:79:10:02,ifname=eth2,obs_id=2,pro
to=1,src_ip=192.168.17.2,src_mac=10:42:17:c1:ce:02
```

```
> SELECT MEAN(packetDeltaCount) FROM domain1 WHERE src_ip='192.168.17.2' AND
dst_ip='192.168.17.1' AND dir='out' AND TIME > now()-5m GROUP BY time(1m);
name: domain1
time                mean
----                ----
1549892460000000000 11
1549892520000000000 10.909090909090908
1549892580000000000 10.818181818181818
1549892640000000000 11
1549892700000000000 10.9
1549892760000000000 11
```

Listing 5.2. Querying InfluxDB for IPFIX data

In order to understand how to use the data stored in the database, please read InfluxDB documentation on:

- schema exploration [28],
- data exploration [29], and
- functions [30].

Each IPFIX domain Y is stored in a dedicated InfluxDB measurement “domainY”. The IP flows are identified using InfluxDB tags listed in Appendix B. The statistics are stored as InfluxDB fields listed in Appendix A. If you need to match IPFIX data with ONOS data, please consult Appendix C.

InfluxDB gives multitude possibilities for querying & aggregating data using the InfluxQL query language. It also provides many software libraries for accessing the data from every popular programming language. If you prefer a graphical tool, then Chronograf [10] is a simple web-based user interface that makes querying InfluxDB easier.

References

1. https://tools.ietf.org/html/rfc7011
2. http://www.openvswitch.org/
3. https://docs.influxdata.com/influxdb/
4. https://www.iana.org/assignments/ipfix/
5. https://docs.influxdata.com/influxdb/v1.7/query_language/data_exploration/
6. https://www.influxdata.com/time-series-platform/
7. https://en.wikipedia.org/wiki/IP_Flow_Information_Export
8. https://en.wikipedia.org/wiki/NetFlow
9. https://tools.ietf.org/html/rfc7012
10. https://docs.influxdata.com/chronograf/v1.7/
11. https://github.com/openvswitch/ovs/blob/master/ofproto/ofproto-dpif-ipfix.c
12. https://github.com/openvswitch/ovs/blob/master/ofproto/ipfix.xml
13. http://www.openvswitch.org/features/
14. https://www.iana.org/assignments/protocol-numbers/
15. https://www.iana.org/assignments/ieee-802-numbers/
16. http://www.openvswitch.org/support/dist-docs/
17. http://www.openvswitch.org/support/dist-docs/ovs-vswitchd.conf.db.5.html
18. http://www.openvswitch.org/support/dist-docs/ovs-vsctl.8.html
19. https://github.com/britram/python-ipfix
20. https://www.ntop.org/products/netflow/nprobe/
21. http://www.pmacct.net/
22. https://github.com/iitis/ipflux
23. https://www.linode.com/docs/networking/ssh/using-gnu-screen-to-manage-persistent-terminal-sessions/
24. https://docs.influxdata.com/influxdb/v1.7/introduction/installation/
25. https://docs.influxdata.com/influxdb/v1.7/administration/upgrading/
26. https://docs.influxdata.com/influxdb/v1.7/guides/downsampling_and_retention/
27. https://docs.influxdata.com/influxdb/v1.7/concepts/key_concepts/
28. https://docs.influxdata.com/influxdb/v1.7/query_language/schema_exploration/
29. https://docs.influxdata.com/influxdb/v1.7/query_language/data_exploration/
30. https://docs.influxdata.com/influxdb/v1.7/query_language/functions/

Appendix A: features available as InfluxDB fields

Below list gives IPFIX traffic features exported by OVS and stored in the InfluxDB database as non-indexed fields. See Appendix B for a subset of features that is indexed.

See the IANA IPFIX document for descriptions of below traffic features [4].

observationPointId	integer
flowDirection	integer
sourceMacAddress	string
destinationMacAddress	string
ethernetType	integer // see [15]
ethernetHeaderLength	integer
ingressInterface	integer
ingressInterfaceType	integer
egressInterface	integer // for egress flows
egressInterfaceType	integer // for egress flows
interfaceName	string // e.g. “eth1”
interfaceDescription	string
ipVersion	integer
ipTTL	integer
protocolIdentifier	integer // see [14]
ipDiffServCodePoint	integer
ipPrecedence	integer
ipClassOfService	integer
sourceIPv4Address	string // for IPv4
destinationIPv4Address	string // for IPv4
sourceIPv6Address	string // for IPv6
destinationIPv6Address	string // for IPv6
flowLabelIPv6	integer // for IPv6
sourceTransportPort	integer // for TCP/UDP/SCTP
destinationTransportPort	integer // for TCP/UDP/SCTP
icmpTypeIPv4	integer // for ICMPv4
icmpCodeIPv4	integer // for ICMPv4
icmpTypeIPv6	integer // for ICMPv6
icmpCodeIPv6	integer // for ICMPv6
flowStartDeltaMicroseconds	integer
flowEndDeltaMicroseconds	integer

droppedPacketDeltaCount	integer
droppedPacketTotalCount	integer
packetDeltaCount	integer
packetTotalCount	integer
ingressUnicastPacketTotalCount	integer
ingressMulticastPacketTotalCount	integer
ingressBroadcastPacketTotalCount	integer
egressUnicastPacketTotalCount	integer
egressMulticastPacketTotalCount	integer
egressBroadcastPacketTotalCount	integer
postMCastOctetDeltaCount	integer
postMCastOctetTotalCount	integer
layer2OctetDeltaCount	integer
layer2OctetTotalCount	integer
flowEndReason	integer
droppedOctetDeltaCount	integer
droppedOctetTotalCount	integer
octetDeltaCount	integer
octetTotalCount	integer
octetDeltaSumOfSquares	integer
octetTotalSumOfSquares	integer
minimumIpTotalLength	integer
maximumIpTotalLength	integer
postMCastPacketDeltaCount	integer
postMCastPacketTotalCount	integer
tcpAckTotalCount	integer
tcpFinTotalCount	integer
tcpPshTotalCount	integer
tcpRstTotalCount	integer
tcpSynTotalCount	integer
tcpUrgTotalCount	integer

Appendix B: features available as InfluxDB tags

Below list gives traffic features (see appendix A) that are indexed in the database as InfluxDB tags, in addition to being available as non-indexed fields. Note the indexed versions are stored in the database under different, shortened identifiers given below.

Take care to query only by tags. Avoid querying by field values (for performance reasons).

time	(automatic)
obs_id	observationPointId
ifname	interfaceName
dir	flowDirection, "in" for 0, "out" for 1
proto	protocolIdentifier (see [14])
src_mac	sourceMacAddress
src_ip	sourceIPv4Address OR sourceIPv6Address
src_port	sourceTransportPort
dst_mac	destinationMacAddress
dst_ip	destinationIPv4Address OR destinationIPv6Address
dst_port	destinationTransportPort

Appendix C: matching IPFIX data with ONOS

Each switch must have a unique numeric identifier “X”: an integer between 1 and 253. If more than one instance of the SerIoT network needs to run on the same infrastructure, then each instance must have a numeric identifier “Y”: an integer between 1 and 253.

When configuring OVS to work under ONOS, do the following:

```
# set OpenFlow datapath ID using Y and X
# left-align the number with zeros, 1->001, 12->012, 123->123
ovs-vsctl set bridge br0 other_config:datapath-id=0000000000YYYXXX

# after that, add the controller, e.g.
ovs-vsctl set-controller br0 tcp:192.168.100.2
```

Listing C.1. Setting OVS OpenFlow datapath identifier.

If OVS is configured properly as explained above and in Section 3.1., then matching IPFIX data with ONOS is simple:

- ONOS switch identifier gives you the IPFIX domain and observation points: domain is the InfluxDB measurement name, observation point is stored in the “obs_id” tag (see Appendix B).
- ONOS flows give MAC addresses, stored in the “src_mac” and “dst_mac” tags (see Appendix B).
- ONOS port number is the suffix of the interface name (e.g. “eth1”), stored in the “ifname” tag (see Appendix B).