Many Uses of Flow and Flow-like Data

Avi Freedman, CEO
NANOG 65
October 2015
Background: NetFlow

• NetFlow is:
  – 20-year old technology now supported in some variant by most network devices.
  – Workable on most common ISP/carrier devices now.
• sFlow came later, is simpler and more accurate in real-time because it’s just packet sampling.
• IPFIX and Netflow v9 are extensible via templates, and allow sending more than just ‘basic flow’ data via those templates.
‘Basic’ Flow

• Basic flow records contain byte and packet counters, TCP Flags, AS, next-hop, and other data aggregated by (usually) the ‘5 tuple’ of (protocol, srcip, dstip, srcport, dstport).

• Most devices support a fixed sampling rate.

• Despite the simplicity of data, there are many use cases for basic flow data for monitoring availability, efficiency, and security of networks, hosts, and applications.
State of Device Export

• sFlow is more common at the switch layer, and NetFlow/IPFIX is more common in routers, but many devices support both protocols.

• Still possible to negatively impact packet forwarding by enabling flow export, but accuracy and stability is generally fine w/ correct software versions. Much, much better than 5+ years ago.
State of Flow Tools

• Flow tools all have some suck. Some suck more and some suck less. No perfect eng+perf+BI+ops tool.
• OSS tools don’t cluster, but popular.
• Most downloadable commercial sw has scale.
• Appliances are either expensive and security-focused, or over-aggregate and can’t support high-res lookback.
• Many tools groups working with Hadoop-ish, Spark, Elastic, and/or live streaming/CEP tools.
• Newer vendors are taking more big-data approach and generally doing private and/or public cloud.
• Extensibility + openness key for augmented flow use cases.
Classic Flow Use Cases

Classic use cases include:

- Congestion analysis for providers and/or customers
- Peering analytics
- Trending, planning and forecasting
- (d)DoS detection (primarily volumetric)
- Basic forensic/historic (who did an IP talk to)
- Modeling of TE, what-if analysis
- Customer cost analysis (Flow + BGP communities)
Classic View: Interface -> Interface Traffic

Bits/s by InterfaceTopTalkers

- TIME OPTIONS
- GROUP BY METRIC
- UNITS
- DATASET

Devices Search

Select: All / None

Selected: 1

Overlay - 0 days

Export | SQL | Add to Dashboard

Graph showing bits/s by InterfaceTopTalkers with overlay options.

Click to select, Shift+Click to multi-select

<table>
<thead>
<tr>
<th>input_port_all</th>
<th>Avg Mb/sec</th>
<th>Percent Total</th>
<th>95th Percentile</th>
<th>Max Mb/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>GigabitEthernet5/7 (263) -&gt; GigabitEthernet1/1 tx:1:23 (1)</td>
<td>56</td>
<td>11.13</td>
<td>62</td>
<td>64</td>
</tr>
<tr>
<td>GigabitEthernet4/17 (209) -&gt; GigabitEthernet1/1 tx:1:23 (1)</td>
<td>50</td>
<td>9.92</td>
<td>61</td>
<td>62</td>
</tr>
<tr>
<td>GigabitEthernet1/1 tx:1:23 (1) -&gt; GigabitEthernet5/17 (273)</td>
<td>49</td>
<td>9.70</td>
<td>63</td>
<td>72</td>
</tr>
<tr>
<td>GigabitEthernet5/17 (273) -&gt; GigabitEthernet1/1 tx:1:23 (1)</td>
<td>45</td>
<td>8.89</td>
<td>63</td>
<td>75</td>
</tr>
</tbody>
</table>
Classic View: Remote Network Analytics
Classic View: Traffic by top AS_PATHs
# Classic View: dDoS Detection

<table>
<thead>
<tr>
<th>Key</th>
<th>Alert Name</th>
<th>Criticality</th>
<th>State</th>
<th>Key Type</th>
<th>Output 1 Name:Value</th>
<th>Output 2 Name:Value</th>
<th>Alert ID</th>
<th>Start</th>
<th>End</th>
<th>Time Over Threshold</th>
<th>Recent Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>many src_ip s_to_1 dst</td>
<td></td>
<td>Major</td>
<td>ACK_REQ</td>
<td>ipv4 dst_a addr</td>
<td>src_ips : 189</td>
<td>pps : 3277</td>
<td>3536</td>
<td>2015-08-26 20:25</td>
<td>2015-08-26 20:46</td>
<td>45%</td>
<td></td>
</tr>
<tr>
<td>high_fps_per_dst_ip ...</td>
<td></td>
<td>Major</td>
<td>ACK_REQ</td>
<td>ipv4 dst_a addr</td>
<td>fps : 110</td>
<td>pps : 118835</td>
<td>3537</td>
<td>2015-08-26 20:25</td>
<td>2015-08-26 20:45</td>
<td>42%</td>
<td></td>
</tr>
<tr>
<td>all_dst 53 or src53_to_1 ip ...</td>
<td></td>
<td>Major</td>
<td>ACK_REQ</td>
<td>ipv4 dst_a addr</td>
<td>pps : 51166</td>
<td>mbps : 576</td>
<td>462</td>
<td>2015-08-26 20:25</td>
<td>2015-08-26 20:44</td>
<td>31%</td>
<td></td>
</tr>
<tr>
<td>udp_src cdst0 ...</td>
<td></td>
<td>Major</td>
<td>ACK_REQ</td>
<td>ipv4 dst_a addr</td>
<td>pps : 86391</td>
<td>mbps : 914</td>
<td>452</td>
<td>2015-08-26 20:25</td>
<td>2015-08-26 20:44</td>
<td>31%</td>
<td></td>
</tr>
<tr>
<td>many src_ip s_to_1 dst</td>
<td></td>
<td>Major</td>
<td>ACK_REQ</td>
<td>ipv4 dst_a addr</td>
<td>src_ips : 137</td>
<td>pps : 13517</td>
<td>3536</td>
<td>2015-08-26 20:37</td>
<td>2015-08-26 20:47</td>
<td>33%</td>
<td></td>
</tr>
</tbody>
</table>
‘Augmented’ Flow

• ‘Who talked to who’ data is great, but if we can get:
  – Semantics (URL, DNS query, SQL query, …)
  – Application performance info (latency, TTFB, …)
  – Network performance info (RTT, loss, jitter, …)

  from passive observation, it unlocks even more/more interesting use cases!

• With many of the same basic report structures.

• Some of this is already available via IPFIX/V9.
Sources of ‘Augmented’ Flow

- **Server-side**
  - OSS sensor software: nprobe, argus
  - Commercial sensors: nBox, nPulse, and others
  - Packet Brokers: Ixia and Gigamon (IPFIX, potentially more)
  - IDS (bro) – a superset of most flow fields, + app decode
  - Web servers (nginx, varnish) – web logs + tcp_info for perf
  - Load balancers – advantage of seeing HTTPS-decoded URLs
  - CISCO AVC, Netflow Lite – generally only on small devices

- Common challenge: Some of the exporters don’t support sampling, and many tools can’t keep up with un-sampled flow.
Augflow Examples: Cisco AVC

docwiki.cisco.com/wiki/AVC-Export:PfR#PfR_NetFlow_Export_CLI

---

Client: Option Active Performance
Exporter Format: NetFlow Version 9
Template ID : 268
Source ID : 0
Record Size : 61
Template layout

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Offset</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow end</td>
<td>153</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>pfr_br_ipv4_address</td>
<td>39000</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>reason id</td>
<td>39002</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>counter_packets_dropped</td>
<td>37000</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>transport_packets_lost_counter</td>
<td>37019</td>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>transport_round-trip-time</td>
<td>37016</td>
<td>24</td>
<td>4</td>
</tr>
<tr>
<td>transport_rtp_jitter_mean</td>
<td>37023</td>
<td>28</td>
<td>4</td>
</tr>
<tr>
<td>mos_worst_100</td>
<td>42115</td>
<td>32</td>
<td>4</td>
</tr>
<tr>
<td>counter_packets_dropped_permanent_short</td>
<td>37001</td>
<td>36</td>
<td>4</td>
</tr>
<tr>
<td>transport_packets_lost_counter_permanent</td>
<td>37020</td>
<td>40</td>
<td>4</td>
</tr>
<tr>
<td>long_term_round-trip-time</td>
<td>39006</td>
<td>44</td>
<td>4</td>
</tr>
<tr>
<td>flow_class_wide</td>
<td>95</td>
<td>48</td>
<td>6</td>
</tr>
<tr>
<td>interface_output_snmp_short</td>
<td>14</td>
<td>54</td>
<td>2</td>
</tr>
<tr>
<td>pfr_status</td>
<td>39001</td>
<td>56</td>
<td>2</td>
</tr>
<tr>
<td>flow_active_timeout</td>
<td>36</td>
<td>58</td>
<td>2</td>
</tr>
<tr>
<td>ip_protocol</td>
<td>4</td>
<td>60</td>
<td>1</td>
</tr>
</tbody>
</table>
augflow Examples: Citrix AppFlow


tcpRTT
The round trip time, in milliseconds, as measured on the TCP connection. This can be used as a metric to determine the client or server latency on the network.

httpRequestMethod
An 8-bit number indicating the HTTP method used in the transaction. An options template with the number-to-method mapping is sent along with the template.

httpRequestSize
An unsigned 32-bit number indicating the request payload size.

httpRequestURL
The HTTP URL requested by the client.
augflow Examples: nTop

http://ntop.org

template.c in nprobe (and elsewhere)

```c
    { 0, BOTH_IPV4_IPV6, FLOW_TEMPLATE, SHORT_SNAPLEN, NTOP_ENTERPRISE_ID, NTOP_BASE_ID+110, STATIC_FIELD_LEN, 4, numeric_format, dump_as_uint, "RETRANSMITTED_OUT_PKTS", "", "Number of retransmitted TCP flow packets (dst->src)" },
    { 0, BOTH_IPV4_IPV6, FLOW_TEMPLATE, SHORT_SNAPLEN, NTOP_ENTERPRISE_ID, NTOP_BASE_ID+101, STATIC_FIELD_LEN, 2, ascii_format, dump_as_ascii, "SRC_IP_COUNTRY", "", "Country where the src IP is located" },
    { 0, BOTH_IPV4_IPV6, FLOW_TEMPLATE, SHORT_SNAPLEN, NTOP_ENTERPRISE_ID, NTOP_BASE_ID+86, STATIC_FIELD_LEN, 4, numeric_format, dump_as_uint, "APPL_LATENCY_SEC", "", "Application latency (sec)" },
    { 0, BOTH_IPV4_IPV6, FLOW_TEMPLATE, SHORT_SNAPLEN, NTOP_ENTERPRISE_ID, NTOP_BASE_ID+82, STATIC_FIELD_LEN, 4, numeric_format, dump_as_uint, "CLIENT_NW_DELAY_SEC", "", "Network latency client <-> nprobe (sec)" },
```
# augflow Examples: nginx, bro

- [https://www.bro.org/sphinx/logs/index.html](https://www.bro.org/sphinx/logs/index.html)

**nginx:** log_format combined '
*remote_addr* - *remote_user* [*time_local]*' "request" $status $body_bytes_sent ' "http_referer" "$http_user_agent"' ‘$tcpinfo_rtt, $tcpinfo_rttvar, $tcpinfo_snd_cwnd, $tcpinfo_rcv_space’;

<table>
<thead>
<tr>
<th>#</th>
<th>cat conn.log</th>
<th>bro-cut</th>
<th>id.orig_h</th>
<th>id.orig_p</th>
<th>id.resp_h</th>
<th>duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>141.142.220.202</td>
<td>5353</td>
<td>224.0.0.251</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>fe80::217:f2ff:fed7:cf65</td>
<td>5353</td>
<td>ff02::fb</td>
<td>141.142.2.2</td>
<td>0.000435</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>141.142.220.50</td>
<td>5353</td>
<td>224.0.0.251</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>141.142.220.118</td>
<td>43927</td>
<td>141.142.2.2</td>
<td>0.000420</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>141.142.220.118</td>
<td>37676</td>
<td>141.142.2.2</td>
<td>0.000392</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>141.142.220.118</td>
<td>40526</td>
<td>141.142.2.2</td>
<td>0.000317</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>141.142.220.118</td>
<td>32902</td>
<td>141.142.2.2</td>
<td>0.000343</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>141.142.220.118</td>
<td>59816</td>
<td>141.142.2.2</td>
<td>0.000375</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>141.142.220.118</td>
<td>59714</td>
<td>141.142.2.2</td>
<td>0.000339</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[...]

```bash
# cat conn.log | bro-cut id.orig_h id.orig_p id.resp_h duration
141.142.220.202 5353 224.0.0.251 -
fe80::217:f2ff:fed7:cf65 5353 ff02::fb -
141.142.220.50 5353 224.0.0.251 -
141.142.220.118 43927 141.142.2.2 0.000435
141.142.220.118 37676 141.142.2.2 0.000420
141.142.220.118 40526 141.142.2.2 0.000392
141.142.220.118 32902 141.142.2.2 0.000317
141.142.220.118 59816 141.142.2.2 0.000343
141.142.220.118 59714 141.142.2.2 0.000375
141.142.220.118 58206 141.142.2.2 0.000339
[...]
```
Storing and Accessing Augmented Flow

• Data back-ends need to be able to understand and ingest the extra fields.
• Often requires integration (for OSS/big data tools) or vendor support.
• And if the tools aren’t ‘open’ via API, SQL, or CLI, data can be trapped and not as useful.
• Many first use cases are ad-hoc to prove effectiveness, then drive to UI reports/dashboards.
• Holy grail: end user app perf + net perf + net flow + host perf + app internals instrumentation.
Extensible Flow Storage: fastbit

- [https://sdm.lbl.gov/fastbit/](https://sdm.lbl.gov/fastbit/)
- [https://github.com/CESNET/ipfixcol/](https://github.com/CESNET/ipfixcol/)
- [http://www.ntop.org](http://www.ntop.org)

(nprobe CLI)

```
fbquery -c
'DST_AS,L4_SRC_PORT,sum(IN_BYTES) as inb,sum(OUT_BYTES) as outb'
-q 'SRC_AS <> 3 AND L4_SRC_PORT <> 80'
-g 'DST_AS,L4_SRC_PORT'
-o 'inb'
-r -L 10 -d .
```
Storing Augmented Flow in Fastbit

root@s5:/data/fb/333/dev1/3/2015/10/03/20/49# ls

APPLATENCY
APPLATENCY.idx
CTIMESTAMP
CTIMESTAMP.idx
DEFAULT_COLUMN
DEFAULT_COLUMN.idx
DEVICE_ID
DEVICE_ID.idx
DNS
DNSQ.idx
DST_AS
DST_AS.idx
DST_GEO
DST_GEO.idx
DST_GEO_CITY
DST_GEO_CITY.idx
DST_GEO_REGION
DST_GEO_REGION.idx
DST_ROUTE_LENGTH
DST_ROUTE_LENGTH.idx
INPUT_PORT
INPUT_PORT.idx
IN_BYTES
IN_BYTES.idx
IN_PKTS
IN_PKTS.idx
IPV4_DST_ADDR
IPV4_DST_ADDR.idx
IPV4_DST_ROUTE_PREFIX
IPV4_DST_ROUTE_PREFIX.idx
IPV4_NEXT_HOP
IPV4_NEXT_HOP.idx
IPV4_SRC_ADDR
IPV4_SRC_ADDR.idx
IPV4_SRC_ROUTE_PREFIX
IPV4_SRC_ROUTE_PREFIX.idx
IPV6_DST_ADDR_HIGH
IPV6_DST_ADDR_HIGH.idx
IPV6_DST_ADDR_LOW
IPV6_DST_ADDR_LOW.idx
IPV6_SRC_ADDR_HIGH
IPV6_SRC_ADDR_HIGH.idx
IPV6_SRC_ADDR_LOW
IPV6_SRC_ADDR_LOW.idx
L4_DST_PORT
L4_DST_PORT.idx
L4_SRC_PORT
L4_SRC_PORT.idx
MPLS_TYPE
MPLS_TYPE.idx
OUTPUT_PORT
OUTPUT_PORT.idx
OUT.Bytes
OUT.Bytes.idx
Use Case: Network Performance

• If the flow system can aggregate by arbitrary dimensions by AS, AS_PATH, Geo, Prefix, etc...
• Then looking at raw network performance from passive sources can be very useful.
• Ex: TCP rexmit by AS_PATH (i.e. from nprobe for a server or, via span/tap, a sensor).
• Important to weight absolute relevance (not just % loss if a few 3 pkt flows).
SQL -> Fastbit Querying for rexmit

Retransmits > .1% by ASN at prime-time for ASNs with > 10k pkts:

```
SELECT i_start_time, src_AS, dst_AS,
sum(tcp_retransmit) AS f_sum_tcp_retransmit,
sum(out_pkts) AS f_sum_out_pkts,
round(((sum(tcp_retransmit)/sum(out_pkts))*1000)/10
AS Perc_retransmits FROM _com WHERE
i_start_time >= '2015-01-09 22:00:00' AND
i_start_time < '2015-01-10 06:00:00' GROUP BY
src_AS, dst_AS, i_start_time HAVING sum(out_pkts) > 10000 AND (sum(tcp_retransmit)/sum(out_pkts))*100 > 0.1 ORDER BY Perc_retransmits DESC;
```
Augmented Flow: rexmit by Dest ASN
Augmented Flow: rexmit by 2\textsuperscript{nd} hop ASN

---

### % Retransmits by dst_second_asn

**Devices Search**

- cat2_cloudhelix.com
- core_nyc_isp.com
- rx1_cloudhelix.com

**Select All / None**

- Selected: 1

**Overlay**

- 0 days

**% Retransmits by dst_second_asn**

**mm01_readnews.com**

---

**Click to select, Shift+Click to multi-select**

<table>
<thead>
<tr>
<th>dst_second_asn</th>
<th># of Retransmits</th>
<th>% of Retransmits</th>
<th>Total Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg/sec</td>
<td>95th percentile</td>
<td>Max/sec</td>
</tr>
<tr>
<td>LGI-UPC Liberty Global Operations B.V., AT (6830)</td>
<td>1.92278</td>
<td>8.51667</td>
<td>8.96667</td>
</tr>
<tr>
<td>COGENT-174 - Cogent Communications, US (174)</td>
<td>0.14361</td>
<td>1.41667</td>
<td>1.41667</td>
</tr>
<tr>
<td>TELEFONICA Telefonica Backbone Autonomous System, ES (12956)</td>
<td>0.00972</td>
<td>0.58333</td>
<td>0.58333</td>
</tr>
</tbody>
</table>

---
Augmented Flow: rexmit by AS_PATH
Use Case: Application-Level Attacks

- With URL and performance data, many kinds of application attacks can be detected.
- To get URL info in an HTTPS world, will need to get data from load balancers or web logs.
- Simplest is WAF – looking for SQL fragments, binary, or other known attack vectors.
- Can hook alerts to mitigation methods, even if running OOB (for example, send TCP FIN/RST in both directions)
Use Case: ‘APM Lite’

• Combining network with application data, you can answer questions like:
  – Show/aggregate cases where application performance is impaired but we know there is no network-layer issue (very useful), and agg by POP, server, app section.
  – Or where there is impairment in both.
  – And ignore network-layer issues where users are unaffected.

• Easy first use case: API perf debugging for web page assembly, or debugging CDN origin pull.
Use Case: Bot detection

• With performance information combined with URL, basic e-commerce bot detection is possible.
• Many attacks are advanced so may require a packet approach to get complete visibility, but basic visibility can often demonstrate a problem.
• Can sometimes be done with syslog analytics, but flow tools often aggregate in interesting ways (geo, AS) that syslog analytics don’t, at least out of the box.
Modern ‘Flow’ Format: kflow

• At today’s speeds, templated formats may not be the most efficient (space/CPU) implementation.
• Working on an open-spec format called kflow with open source tools to take to and from NetFlow, sFlow, IPFIX, nginx and bro logs, and Cisco, Citrix, ntop, and other vendor formats.
• Based on Cap’n Proto, which is a ‘serialization’ lib that is basically a struct with 0-packing - https://capnproto.org/
• Drawback: Can’t delete fields, just 0-pack them.
• Will shortly be live at https://github.com/Kentik
Flow with Cap’n Proto

struct kflow_v1 {
  version @44: Int64;
  timestampNano @0: Int64;
  dstAs @1: UInt32;
  dstGeo @2: UInt32;
  dstMac @3: UInt32;
  headerLen @4: UInt32;
  inBytes @5: UInt64;
  inPkts @6: UInt64;
  inputPort @7: UInt32;
  ipSize @8: UInt32;
  ipv4DstAddr @9: UInt32;
  ipv4SrcAddr @10: UInt32;
  tcpRetransmit @27: UInt32;
  dstBgpAsPath @34: Text;
  dstBgpCommunity @35: Text;
  <...>
Comments / Questions?

Avi Freedman
avi (at) kentik.com