Flowtuples IV: Reality Strikes Back

Shane Alcock
Alcock Network Intelligence Ltd
shane@alcock.co.nz

DUST 2021

What are flowtuples?

- Pcaps from the UCSD-NT are massive
  - Processing takes a long time, storage fills up quickly
- Solution: aggregate similar packets into “flowtuples”
  - Retain key header fields
  - Group multiple equivalent packets into a single record
- Regular flowtuple data has been collected from UCSD-NT since 2008

Flowtuple v3

- STARDUST gave us an opportunity to redesign flowtuples
  - Add useful meta-data to flowtuple records
  - Save records in a mainstream big data format (Avro)
  - Create tools / APIs for simplifying large-scale flowtuple analysis

Flowtuple v3

src_ip	dst_ip	src_port	dst_port	protocol	ttl	tcp_flags	ip_len	packet_cnt
1.2.3.4	10.100.0.45	44120	22	6	122	2 (SYN)	52	4

tcp_synlen	tcp synwinlen	is_spoofed	is_masscan	maxmind continent	maxmind country	netacq continent	netacq country	prefix2asn
32	65535	False	False	AS	KR	AS	KR	123456

Flowtuple v3

- File storage structure
  - Flowtuples are written to disk every minute
  - Files are rotated every minute as well
  - Directly uploaded into Swift object store

- Deployed in mid-2020

And all was well...

Reality Ensues

Reality Ensues

- Storage requirements were clearly not sustainable
  - 160 TB per year, without factoring in traffic growth over time
    - 300 MB per minute in 2020
    - 40 MB per minute in 2015
    - 1 MB per minute in 2008

- Back to the drawing board...

What **really** matters

- Specific destination IPs in dark space DO NOT matter
  - Are these packets hitting a small number of addresses, or many?
  - Are these packets hitting a wide range of subnets, or just a few?

- Consider a scan of an entire /16 subnet by a single host
  - In FT3, this will result in 65536 near-identical flow records
  - We only need 1 record that includes the number of addresses scanned

What **really** matters

- Source ports for unsolicited traffic are generally ephemeral
  - It might be interesting if the source port is consistent
  - If the source port is inconsistent, that may also be interesting
    - But the specific inconsistent values are NOT interesting

- We can apply the same logic to TTLs, packet sizes and TCP flags
  - Is the value consistent? If so, what are those values?

Flowtuple v4

- Aggregate away the unimportant details…
  - **Destination subnets (/16s)** instead of destination IP addresses
  - Source port, TTL, packet size and TCP flags no longer part of flow key
  - Instead, record "**common**" values for the above
    - Also the total **number of unique values** seen for the flow

- Keep most of the interesting meta-data added in Flowtuple v3

Flowtuple v4

src_ip	dst_net	dst_port	protocol	packet_cnt	uniq_pkt_sizes	uniq_ttls	uniq_dst_ips
1.2.3.4	10.100.0.0	22	6	518	7	1	232

uniq_src ports	uniq_tcp flags	first_syn length	first_tcp rwin	maxmind continent	maxmind country	netacq continent	netacq country	prefix2asn
475	1	32	65535	AS	KR	AS	KR	123456

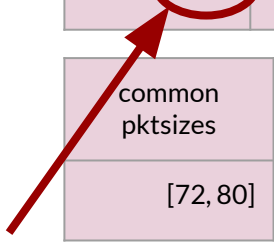
common pktsizes	common pktsize_freqs	common ttls	common ttl_freqs	common srcports	common srcport_freqs	common tcpflags	common tcpflag_freqs
[72, 80]	[356, 141]	[122]	[518]	[ ]	[ ]	[2]	[518]

Flowtuple v4

src_ip	dst_net	dst_port	protocol	packet_cnt	uniq_pkt_sizes	uniq_ttls	uniq_dst_ips
1.2.3.4	10.100.0.0	22	6	518	7	1	232

uniq_src ports	uniq_tcp flags	first_syn length	first_tcp rwin	maxmind continent	maxmind country	netacq continent	netacq country	prefix2asn
475	1	32	65535	AS	KR	AS	KR	123456

common pktsizes	common pktsize_freqs	common ttls	common ttl_freqs	common srcports	common srcport_freqs	common tcpflags	common tcpflag_freqs
[72, 80]	[356, 141]	[122]	[518]	[ ]	[ ]	[2]	[518]



Flowtuple v4

- Extended aggregation time period to 5 minutes
  - Avoid repetitive data storage at the cost of some time precision

- Storage costs:
  - 42 MB per minute avg
  - 80 MB per minute worst case
  - ~25 TB per year

Flowtuple v4

- Updating existing tools to understand FT4
  - pyavro-stardust
    - Python module for reading FT4 files
  - pyspark-stardust
    - Helper routines for processing FT4 with Apache Spark
  - ftconvert
    - Converts older flowtuple files to FT4

Quick Questions?

- https://github.com/CAIDA/corsaro3/wiki/Flowtuple-Formats

Flowtuple Analysis Using Apache Spark

Shane Alcock
Alcock Network Intelligence Ltd
shane@alcock.co.nz

DUST 2021

Apache Spark (simplified)

- Ingredients
  - A cluster full of compute power (running Spark on each node)
  - A lot of data in a supported data store
  - An analysis program written with the Spark API

- Spark does the rest for you

pyspark-stardust

- An extra Python API on top of Spark
  - Implements common methods for exploring STARDUST data
  - Still a work in progress
    - Flowtuple only
  - Aim is to make it very simple to write Spark code for exploring FTs

Example Flowtuple Analysis

- What are the **top 5 countries** responsible for flows where **TTL >= 200**?
- Fetch flowtuples where the source IP matches the prefix **130.0.0.0/8**
  - Generate a **time series of packets** observed from that prefix
- What proportion of flows from **Russia** targeted at least **60,000 unique destination addresses** in a /16?

First Steps

```
import stardust

# the arguments describe where to find the flowtuple files in swift
sd = stardust.StardustPySparkHelper(
        "telescope-ucsdnt-avro-flowtuple-v4-2021 ",
        "ucsd-nt", "v3_5min")

# 4 is the number of CPUs to allocate to the processing job
sd.startSparkSession("dust2021demo", 4)

fts = sd.getFlowtuplesByTimeRange(1612130100, 1612130100 + 3600)
```

First Steps

```
fts = sd.getFlowtuplesByTimeRange(1612130100, 1612130100 + 3600)

print(fts.count())
```

```
357723597
```

First Steps

```
fts = sd.getFlowtuplesByTimeRange(1612130100, 1612130100 + 3600)

for f in fts.limit(10).collect():
    print(f)
```

```
Row(time=1613239200, src_ip=16824709, dst_net=738263040, dst_port=771, protocol=1,
packet_cnt=2, uniq_dst_ips=1, uniq_pkt_sizes=2, uniq_ttls=1, uniq_src_ports=0,
uniq_tcp_flags=0, first_syn_length=0, first_tcp_rwin=0, common_pktsizes=[],
common_pktsize_freqs=[], common_ttls=[44], common_ttl_freqs=[2], common_srcports=[],
common_srcport_freqs=[], common_tcpflags=[], common_tcpflag_freqs=[],
maxmind_continent='??', maxmind_country='??', netacq_continent='AS',
netacq_country='TH', prefix2asn=23969)
... 9 more rows here ...
```

Example Flowtuple Analysis

- What are the **top 5 countries** responsible for flows where **TTL >= 200**?
- Fetch flowtuples where the source IP matches the prefix 130.0.0.0/8
  - Generate a time series of packets observed from that prefix
- What proportion of flows from Russia targeted at least 60,000 unique destination addresses in a /16?

Filtering by Common Values

```
q_res = sd.filterFlowtuplesByCommonValue(fts, "common_ttls",
        [(200, 255)])

print(q_res.count())
for f in q_res.limit(10).collect():
    print(f)
```

```
87020298
Row(time=1613239200, src_ip=16966307, dst_net=738525184, dst_port=2048, protocol=1,
packet_cnt=1, uniq_dst_ips=1, uniq_pkt_sizes=1, uniq_ttls=1, uniq_src_ports=0,
uniq_tcp_flags=0, first_syn_length=0, first_tcp_rwin=0, common_pktsizes=[28],
common_pktsize_freqs=[1], common_ttls=[234], common_ttl_freqs=[1],
common_srcports=[], common_srcport_freqs=[], common_tcpflags=[],
common_tcpflag_freqs=[], maxmind_continent='??', maxmind_country='??',
netacq_continent='AS', netacq_country='TH', prefix2asn=23969)
... 9 more rows here ...
```

Filtering by Common Values

```
q_res = sd.filterFlowtuplesByCommonValue(fts, "common_ttls",
        [(200, 255)])

print(q_res.count())
for f in q_res.limit(10).collect():
    print(f)
```

```
87020298
Row(time=1613239200, src_ip=16966307, dst_net=738525184, dst_port=2048, protocol=1,
packet_cnt=1, uniq_dst_ips=1, uniq_pkt_sizes=1, uniq_ttls=1, uniq_src_ports=0,
uniq_tcp_flags=0, first_syn_length=0, first_tcp_rwin=0, common_pktsizes=[28],
common_pktsize_freqs=[1], common_ttls=[234], common_ttl_freqs=[1],
common_srcports=[], common_srcport_freqs=[], common_tcpflags=[],
common_tcpflag_freqs=[], maxmind_continent='??', maxmind_country='??',
netacq_continent='AS', netacq_country='TH', prefix2asn=23969)
... 9 more rows here ...
```

Getting the Top 5 Countries

```
topn = sd.getTopValuesByFlowCount(q_res, "netacq_country", 5, True)

for k, v in topn.items():
    print(v)
```

```
{'name': 'RU', 'rank': 1, 'flows': 36296567, 'pct': 0.4171, 'cumpct': 0.4171}
{'name': 'SC', 'rank': 2, 'flows': 10111708, 'pct': 0.1161, 'cumpct': 0.5333}
{'name': 'US', 'rank': 3, 'flows': 5688913, 'pct': 0.0653, 'cumpct': 0.5986}
{'name': 'NL', 'rank': 4, 'flows': 5365167, 'pct': 0.0616, 'cumpct': 0.6603}
{'name': 'CY', 'rank': 5, 'flows': 3355182, 'pct': 0.0385, 'cumpct': 0.6988}
{'name': 'Other', 'rank': 6, 'flows': 26202761, 'pct': 0.3011, 'cumpct': 1.0}
```

Example Flowtuple Analysis

- What are the top 5 countries responsible for flows where TTL >= 200?
- Fetch flowtuples where the source IP matches the prefix **130.0.0.0/8**
  - Generate a **time series of packets** observed from that prefix
- What proportion of flows from Russia targeted at least 60,000 unique destination addresses in a /16?

Filter by Prefix → Time Series

```
q_res = sd.filterFlowtuplesByPrefix(fts, u"130.0.0.0/8")

series = sd.generateSeriesFromFlowtuples(q_res, "dustdemo", "130net")
for v in series.collect():
    print(v)
```

```
Row(time=1613239200, pkt_cnt=16716, count(src_ip)=313, count(dst_net)=178,
count(prefix2asn)=53, libts_key='dustdemo.130net')
Row(time=1613239500, pkt_cnt=15138, count(src_ip)=323, count(dst_net)=178,
count(prefix2asn)=53, libts_key='dustdemo.130net')
Row(time=1613239800, pkt_cnt=16785, count(src_ip)=302, count(dst_net)=178,
count(prefix2asn)=55, libts_key='dustdemo.130net')
Row(time=1613240100, pkt_cnt=15880, count(src_ip)=304, count(dst_net)=178,
count(prefix2asn)=54, libts_key='dustdemo.130net')
... 8 more rows ...
```

Example Flowtuple Analysis

- What are the top 5 countries responsible for flows where TTL >= 200?
- Fetch flowtuples where the source IP matches the prefix 130.0.0.0/8
  - Generate a time series of packets observed from that prefix
- What proportion of flows from **Russia** targeted at least **60,000 unique destination addresses** in a /16?

Intersections

```
sect, base = sd.createFlowtupleIntersection(fts,
    ["netacq_country == 'RU' ", "uniq_dst_ips >= 60000 "])

print(sect.count(), base.count())
for f in sect.limit(5).collect():
    print(f)
```

```
3 40293940
Row(time=1613239800, src_ip=3239906926, dst_net=738197504, dst_port=5060,
protocol=17, packet_cnt=64942, uniq_dst_ips=64942, uniq_pkt_sizes=13, uniq_ttls=1,
uniq_src_ports=1, uniq_tcp_flags=0, first_syn_length=0, first_tcp_rwin=0,
common_pktsizes=[], common_pktsize_freqs=[], common_ttls=[49],
common_ttl_freqs=[64942], common_srcports=[5181], common_srcport_freqs=[64942],
common_tcpflags=[], common_tcpflag_freqs=[], maxmind_continent='??',
maxmind_country='??', netacq_continent='EU', netacq_country='RU', prefix2asn=35478)
... 2 more rows ...
```

Intersections

```
sect, base = sd.createFlowtupleIntersection(fts,
    ["netacq_country == 'RU' ", "uniq_dst_ips >= 60000 "])

print(sect.count(), base.count())
for f in sect.limit(5).collect():
    print(f)
```

```
3 40293940
Row(time=1613239800, src_ip=3239906926, dst_net=738197504, dst_port=5060,
protocol=17, packet_cnt=64942, uniq_dst_ips=64942, uniq_pkt_sizes=13, uniq_ttls=1,
uniq_src_ports=1, uniq_tcp_flags=0, first_syn_length=0, first_tcp_rwin=0,
common_pktsizes=[], common_pktsize_freqs=[], common_ttls=[49],
common_ttl_freqs=[64942], common_srcports=[5181], common_srcport_freqs=[64942],
common_tcpflags=[], common_tcpflag_freqs=[], maxmind_continent='??',
maxmind_country='??', netacq_continent='EU', netacq_country='RU', prefix2asn=35478)
... 2 more rows ...
```

Quick Questions?

- https://github.com/CAIDA/stardust-tools/tree/master/pyspark

Flowtuple v1

- Lost to the ravages of time (?)

Flowtuple v2

- Produced by the “old” corsaro software
  - 2008 - 2020

- Custom binary format developed by Alistair King
  - Space-efficient
  - Requires libcorsaro to read flowtuple files
  - Not compatible with modern big data engines

Flowtuple v2

- Data format

src_ip	dst_ip	src_port	dst_port	protocol	ttl	tcp_flags	ip_len	packet_cnt
1.2.3.4	10.100.0.45	44120	22	6	122	2 (SYN)	52	4

Flowtuple v2

- Other key aspects:
  - Reporting interval is 1 minute
  - File rotation interval is hourly
  - Storage cost:
    - ~300 MB per minute in 2020
    - ~40MB per minute in 2015
    - ~1MB per minute in 2008

Apache Spark

- Unified analytics engine for big data processing
  - Automatically divide and distribute tasks across a cluster
  - Task results are combined and returned to the Spark user
  - Programming APIs in several languages
  - SQL language support for expressing queries