About Me

• Mehmet Öner Yalçın
• Senior Network Data Scientist SKY UK Analytics & Planning Team
• Mix of Data Scientist, DevOps and Network Engineer
• Spending some of his time on data/statistical analysis on both network and non network data
• Building tools for network planning, performance measurement and network automation
• Enjoys coding in R, Python and SQL.
What is this talk about?

• It’s not about automating network devices.
• It’s not only about programming.
• It’s not only about data analysis either.

• It is about network measurement using basic tools like ping etc...
• It is about combining open source tools to build a platform for scalable, distributed network performance measurement tool.
What is Hubble?

• If you assume Internet is the universe, you can point Hubble to a certain part of universe (internet) and take measurements.
• Hubble is an automated network layer measurement tool:
  – Concurrently measuring multiple paths to same destination
  – Custom metadata enrichments: BGP, Geolocation
  – Automatic host discovery
  – Scalable up to ~100K concurrent measurements with single probe (Potentially much more with multiple probers)
• Basically a Python library. It is a thin wrapper around network scan, measurement and enrichment tools
• Uses ELK stack (ElasticSearch, Logstash, Kibana) as data store, computation and visualization.
Hubble Story

• Sky peers with a few transit providers to route its traffic globally
• Our transit providers vary in their size and geographic presence
• We did not have any proper performance metric to decide which transit provider is a better choice from Sky’s perspective
• Hubble was created to address this problem.
• Hubble queries subnets of interest, scans hosts in these subnets and measures network layer performance for each Transit Provider.
• We are expanding to do more targeted measurements across Europe.
Problems

• Bad news, measuring transit provider means, measuring whole internet
  – There are more than 700K routes in BGPv4 global routing table, many many hosts.

• Zipf’s Law \( P_n \propto 1/n^a \) says that the frequencies \( f \) of certain events are
  inversely proportional to their rank \( r \). In our case it’s not frequency but
  traffic amount received from each subnet.

• Most of internet traffic is mostly generated from relatively small number
  of, not all 700K subnets are equal. Our netflow analysis shows that 500
  subnets are responsible more than 70% of our transit traffic.

• \(~700K \rightarrow 500\) is a good compromise.

• Hubble queries netflow collectors to find out subnets of interest.
Problems

• How can we ensure egressing (hard requirement) and ingressing traffic (soft requirement) uses the same transit provider?

• Standard routing is destination based. Sourced based routing at P&T routers ensures egressing traffic egresses through the desired transit provider.

• If we require returning traffic to arrive from the same transit provider, we can do it by BGP advertisements.
  – Each group of probers lives in a certain /24 public subnet and this subnet is only advertised from a specific transit provider. This will ensure traffic will ingress from the same egress path.
Routing

TP-1 Prober
subnet A

TP-2 Prober
subnet B

TP-3 Prober
subnet C

Peer & Transit Router

Advertise subnet A to TP-1
Advertise subnet B to TP-2
Advertise subnet B to TP-3

Source route subnet A to TP-1
Source route subnet B to TP-2
Source route subnet B to TP-3
System Requirements

- **Target Selection**: Measurement targets should be chosen automatically and dynamically.

- **Scaling**: Each component can be horizontally scaled if more scaling is required.

- **Microservices**: Each component must be independent, no tight coupling allowed (except ELK stack). Communication should happen through message queues.

- **Development**: Each component should be treated as a plugin, allowing a modular design, for example, adding a TCP probe must be easy and should not require significant integration effort.

- **Enrichment**: Measurement data must be enriched via few resources like BGP and Geo tagging.

- **Data Analysis**: Data ingestion, storage and compute platform must be horizontally scalable and should allow real-time analysis.

- **Data Visualization**: Dashboard should allow users to query data in different dimensions, fast and near real-time.
Components – Subnet ingest

• Hubble takes a list of subnets as input. Currently supports:
  – Arbor API
  – Kentik API
  – Static file

• These subnets are sent for host discovery to message bus.
Components – Host Discovery

• Hubble scans and discovers responding hosts for each subnet. Currently supports:
  – ICMP scans
  – TCP SYN scans

• Under the hood it uses Zmap for scans. [https://zmap.io/](https://zmap.io/)
  – ZMap is a fast single packet network scanner designed for Internet-wide network surveys.

• *We run scans each day. Please see scanning best practices for good internet citizenship*
Components – Host Discovery

```python
from hubble.collect.zmap import Zmap

SKY_PREFIX = '5.64.0.0/13'

zmap = Zmap()  # Create a Zmap instance
# return max 50 Hosts from SKY_PREFIX, scanning bandwidth max is 1Mbps
hosts = zmap.icmp_scan('5.64.0.0/13', max_results=50, bandwidth=1000)

print(hosts)
['5.64.118.25',
 '5.64.154.81',
 '5.64.160.228',
 '5.64.166.63',
 ...,
 '5.71.233.19',
 '5.71.236.232',
 '5.71.70.179']
```
Components – Probers

• Each prober takes measurement from discovered host each configured interval.

• Hubble uses Scamper for taking large scale measurements.
  – Scamper is a tool that actively probes the Internet in order to analyse topology and performance. It is released by Center for Applied Internet Data Analysis (CAIDA).
  – Scamper is designed to actively probe destinations in parallel (at a specified packets-per-second rate) so that bulk data can be collected in a timely fashion.
Components – Probers

```python
from hubble.probers.scamper import Scamper

# provider is a custom tag appened to each scan
prober_a = Scamper(provider='transit_provider_a')

# Ping all in `hosts` list 10 times with 3 seconds interval,
# do not summarize and return as dict, (other option pandas df)
result = prober_a.get_measurement_icmp(host_list=hosts,
                                       interval=3,
                                       probe_count=10,
                                       only_stats=False)
```
{'asn': 5607,
 'dst': '5.71.80.156',
 'ping_sent': 10,
 'provider': 'transit_provider_a',
 'responses': [{
 'reply_ttl': 55,
 'rtt': 12.877,
 'rx': {'sec': 1523823546, 'usec': 353747},
},
 {\'reply_ttl\': 55, \'rtt\': 13.532, \'rx\': {\'sec\': 1523823549, \'usec\': 356475}},
 {\'reply_ttl\': 55, \'rtt\': 17.446, \'rx\': {\'sec\': 1523823552, \'usec\': 365542}},
 {\'reply_ttl\': 55, \'rtt\': 13.023, \'rx\': {\'sec\': 1523823555, \'usec\': 364953}},
 {\'reply_ttl\': 55, \'rtt\': 12.897, \'rx\': {\'sec\': 1523823558, \'usec\': 366755}},
 {\'reply_ttl\': 55, \'rtt\': 12.677, \'rx\': {\'sec\': 1523823561, \'usec\': 369753}},
 {\'reply_ttl\': 55, \'rtt\': 12.494, \'rx\': {\'sec\': 1523823564, \'usec\': 373804}},
 {\'reply_ttl\': 55, \'rtt\': 12.604, \'rx\': {\'sec\': 1523823567, \'usec\': 376804}},
 {\'reply_ttl\': 55, \'rtt\': 12.826, \'rx\': {\'sec\': 1523823570, \'usec\': 380906}},
 {\'reply_ttl\': 55, \'rtt\': 12.788, \'rx\': {\'sec\': 1523823573, \'usec\': 384819}},{\'reply_ttl\': 55, \'rtt\': 13.316, \'avg\': 13.316,
 'loss': 0, \'min\': 12.494,
 'max': 17.446,
 'replies': 10, \'replies\': 10,
 'stddev': 1.402},
 'subnet': '5.64.0.0/13'}
# Components – Probers

```
# Ping all in hosts list 10 times with 3 seconds interval, summarize and
# save as pandas dataframe (default dict, no summarize)
result = prober_level3.get_measurement_icmp(host_list=hosts, interval=3,
                                          probe_count=10, frmt='pandas',
                                          only_stats=True)
```

<table>
<thead>
<tr>
<th>#</th>
<th>asn</th>
<th>avg</th>
<th>dst</th>
<th>loss</th>
<th>max</th>
<th>min</th>
<th>ping_sent</th>
<th>provider</th>
<th>replies</th>
<th>stddev</th>
<th>subnet</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5607</td>
<td>22.271</td>
<td>5.64.119.13</td>
<td>0</td>
<td>50.001</td>
<td>12.628</td>
<td>10</td>
<td>level3</td>
<td>10</td>
<td>10.741</td>
<td>5.64.0.0/13</td>
<td>2018-04-15 21:14:34</td>
</tr>
<tr>
<td>1</td>
<td>5607</td>
<td>13.069</td>
<td>5.64.136.32</td>
<td>0</td>
<td>38.211</td>
<td>8.683</td>
<td>10</td>
<td>level3</td>
<td>10</td>
<td>8.775</td>
<td>5.64.0.0/13</td>
<td>2018-04-15 21:14:34</td>
</tr>
<tr>
<td>2</td>
<td>5607</td>
<td>16.515</td>
<td>5.64.210.52</td>
<td>0</td>
<td>20.955</td>
<td>15.015</td>
<td>10</td>
<td>level3</td>
<td>10</td>
<td>1.955</td>
<td>5.64.0.0/13</td>
<td>2018-04-15 21:14:34</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>5607</td>
<td>10.975</td>
<td>5.71.228.145</td>
<td>0</td>
<td>20.009</td>
<td>9.550</td>
<td>10</td>
<td>level3</td>
<td>10</td>
<td>3.022</td>
<td>5.64.0.0/13</td>
<td>2018-04-15 21:14:36</td>
</tr>
<tr>
<td>48</td>
<td>5607</td>
<td>14.104</td>
<td>5.71.78.43</td>
<td>0</td>
<td>15.370</td>
<td>12.936</td>
<td>10</td>
<td>level3</td>
<td>10</td>
<td>0.670</td>
<td>5.64.0.0/13</td>
<td>2018-04-15 21:14:36</td>
</tr>
<tr>
<td>49</td>
<td>5607</td>
<td>14.035</td>
<td>5.71.80.156</td>
<td>0</td>
<td>18.024</td>
<td>12.742</td>
<td>10</td>
<td>level3</td>
<td>10</td>
<td>1.729</td>
<td>5.64.0.0/13</td>
<td>2018-04-15 21:14:36</td>
</tr>
</tbody>
</table>
Components – Enrichers

• Measurement data is enriched with BGP and GeoLocation metadata so more insights can be inferred.

• ASN information appended to each scan using `pyasn` *(Now Logstash 6 also supports ASN Lookup)*
  – `pyasn` is a Python extension module that enables very fast IP address to Autonomous System Number lookups developed by Economics of Cybersecurity research group at Delft University of Technology

• Elastic’s Logstash module appends GeoLocation tag for each scan before sending to ElasticSearch for indexing.
Components – BigData Compute, Storage and Visualization

• Measurement data saved to **ElasticSearch** NoSQL DB.
  – Elasticsearch is a distributed, RESTful search and analytics engine and horizontally scalable.
  – Elasticsearch lets us perform and combine many types of searches; structured, unstructured, geo and metric.

• Kibana is used as visualization and dashboarding platform.
  – Kibana is used for visualizing Elasticsearch data and navigate the Elastic Stack
Components – Message Bus

• Data moved between instances of components should be done by a message bus.

• Hubble library has support for rabbitmq, this allows each component to scale independently.
  – For example we may need 10 probers to work in parallel, and each prober subscribes to work queue from Zmap. Message bus is responsible for allocating jobs to each prober instance, allowing parallelism and load sharing.
Software Components

Transit Provider-1 prober
- scamper
- elasticsearch
- zmap
- rabbitmq

Subnet Getter
From Netflow (Arbor, Kentik)

Transit Provider-2 prober
- scamper
- logstash (filebeats)
- elasticsearch

Transit Provider-3 prober
- scamper
- logstash (filebeats)
- elasticsearch
Dashboard: Filter by Country

China

Average SD - by Country

0.338 236.619 0.05

50th percentile of Standard Deviation of RTT 50th percentile of RTT Average Packet Loss
Dashboard: Filter by Transit Provider

RTT SD vs Provider
- 50th percentile of Ping RTT Std Deviation in Percentiles
- 95th percentile of Ping RTT Std Deviation in Percentiles
- 99th percentile of Ping RTT Std Deviation in Percentiles

Median STD by Time
- gtt

Median RTT by Time
- gtt

SD by Country

Republic of Lithuania

Average SD - by Country

50th percentile of Standard Deviation of RTT
50th percentile of RTT
Average Packet Loss

0.212  27.182  0.029
Limelight Case

- Limelight is one of the CDNs and similar to other CDNs SKY exchanges important amount of traffic with Limelight. Therefore Limelight ASN always comes up in our target list.
- The Hubble analysis shows Limelight is not well connected via GTT or NTT.
- Median RTT for GTT and NTT is around ~80ms and for Level3 ~12ms.
• Digging deeper shows that GTT is connected to Limelight through their PNI in New York
• NTT does not have direct peering with Limelight so it connects through GTT
Limelight Case – Level 3

- Level 3 has peering with Cogent and Cogent also peers with Limelight in Europe, thus having the lowest RTT
Limitations

- L4 and L7 measurement probes need to be developed for more service layer analysis (currently there is support for only ICMP measurements)
- No support for IPv6 yet
- Hubble forks a new process for Scamper using python’s subprocess module. Hubble wraps only command line interface for Zmap and Scamper. There is no deep level of integration. Needs better integration for parallelism.
- Ping, traceroute ..etc are blocking processes. One layer of parallelism is provided by monkey patching using Python gevent library for subprocess module. asyncio was introduced in Python 3 haven’t exploited it yet.
- Installation is slightly tricky due to dependencies, need to containerize.