Hubble

Internet Telescope

Mehmet Öner Yalçın

About Me

- Mehmet Öner Yalçın
- DevOps Network Planner at SKY UK Analytics & Planning Team
- Mix of Data Analyst, 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.

 /oneryalcin  */oneryalcin*  */oneryalcin*

What is this talk about?

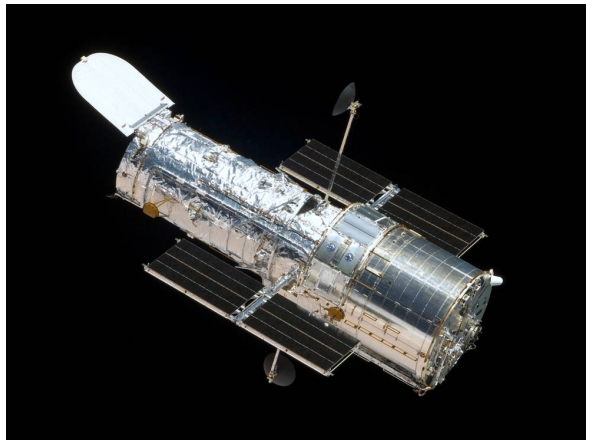
- It's not about automating network devices.
- It's not only about programming.
- It's not only about data analysis either.

- 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.

Edwin Hubble *Credits: NASA*



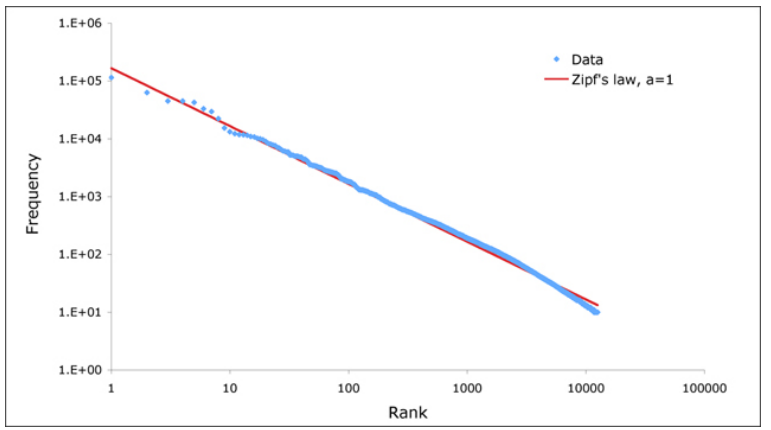
Hubble Space Telescope
Credits: NASA

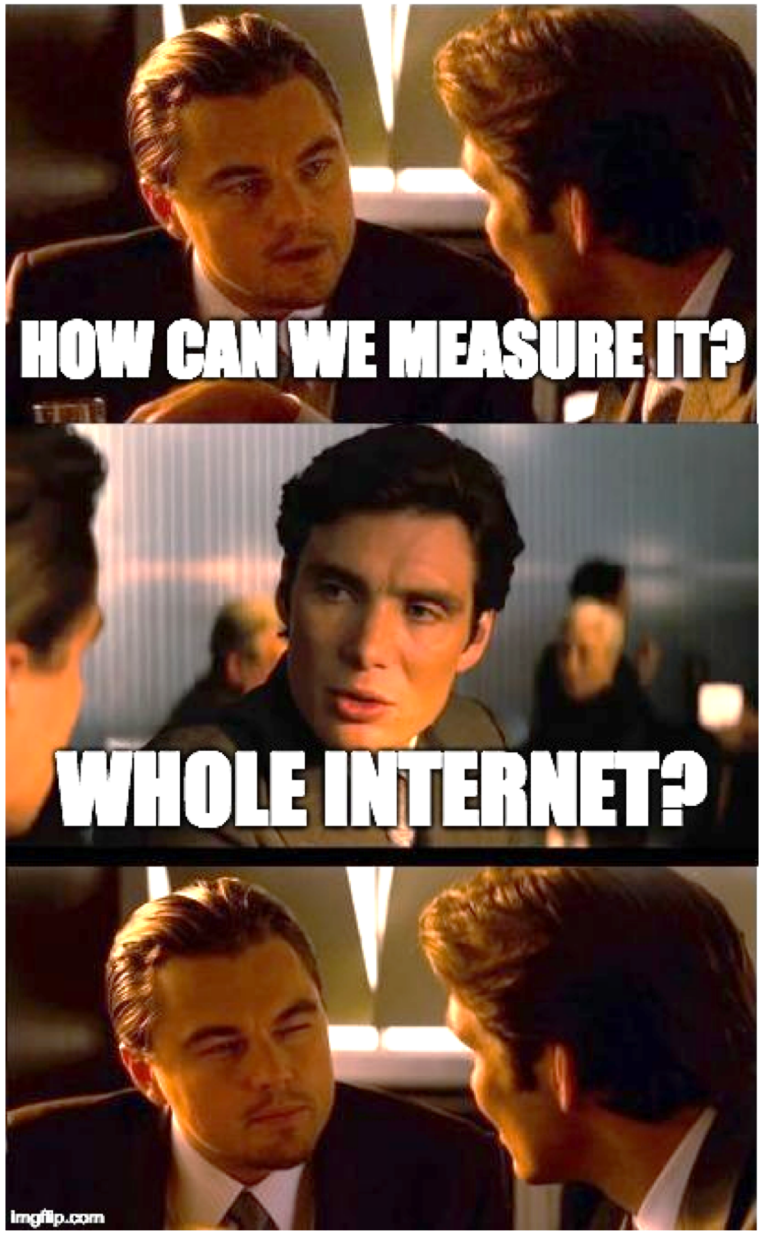
Hubble Story

- 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 addresses 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

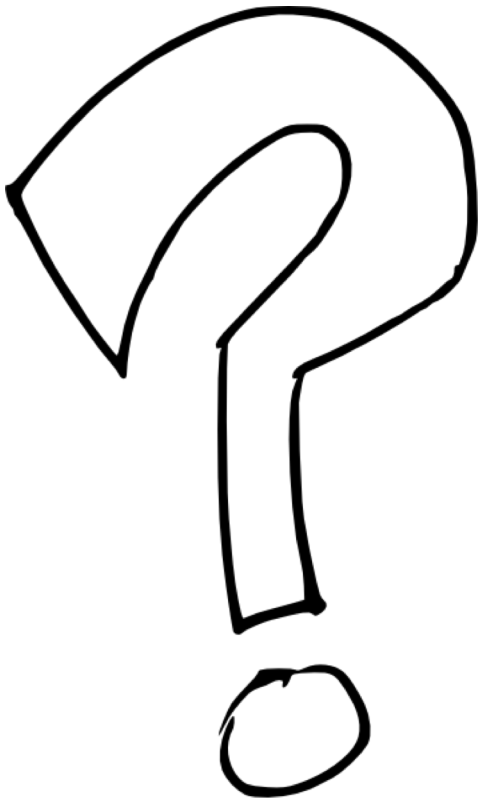
- Bad news, measuring transit provider means, measuring whole internet
  - There are more than 700K routes in BGPv4 global routing table, many many hosts.
- **Zipf's Law** $P_n \propto 1/n^a$ says that the frequencies *f* of certain events are inversely proportional to their rank *r*. In our case it's 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 -> 500 is a good compromise.
- Hubble queries netflow collectors to find out *subnets of interest*.



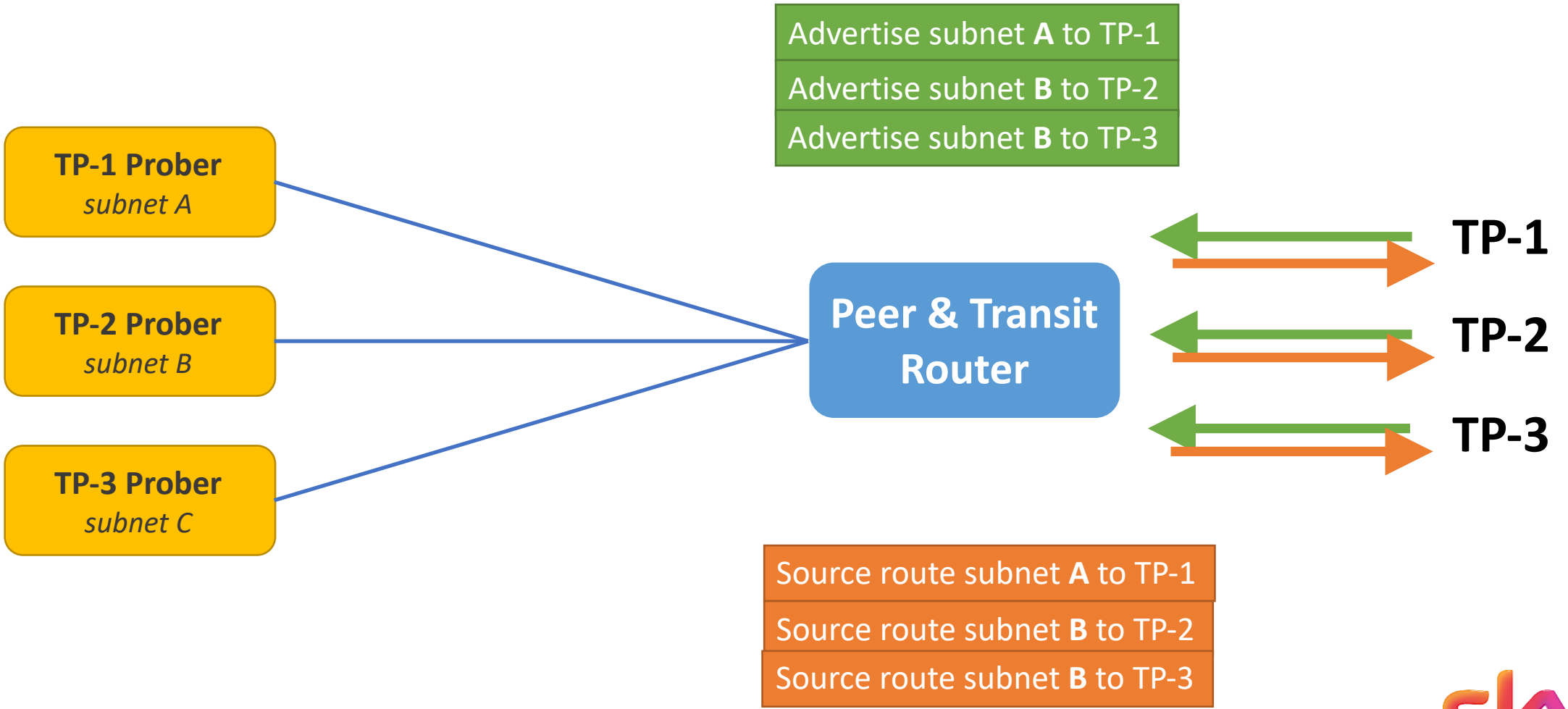


Problems

- 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


Advertise subnet **A** to TP-1
Advertise subnet **B** to TP-2
Advertise subnet **B** to TP-3
TP-1 Prober
subnet A
TP-2 Prober
subnet B
TP-3 Prober
subnet C
Peer & Transit Router
TP-1
TP-2
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

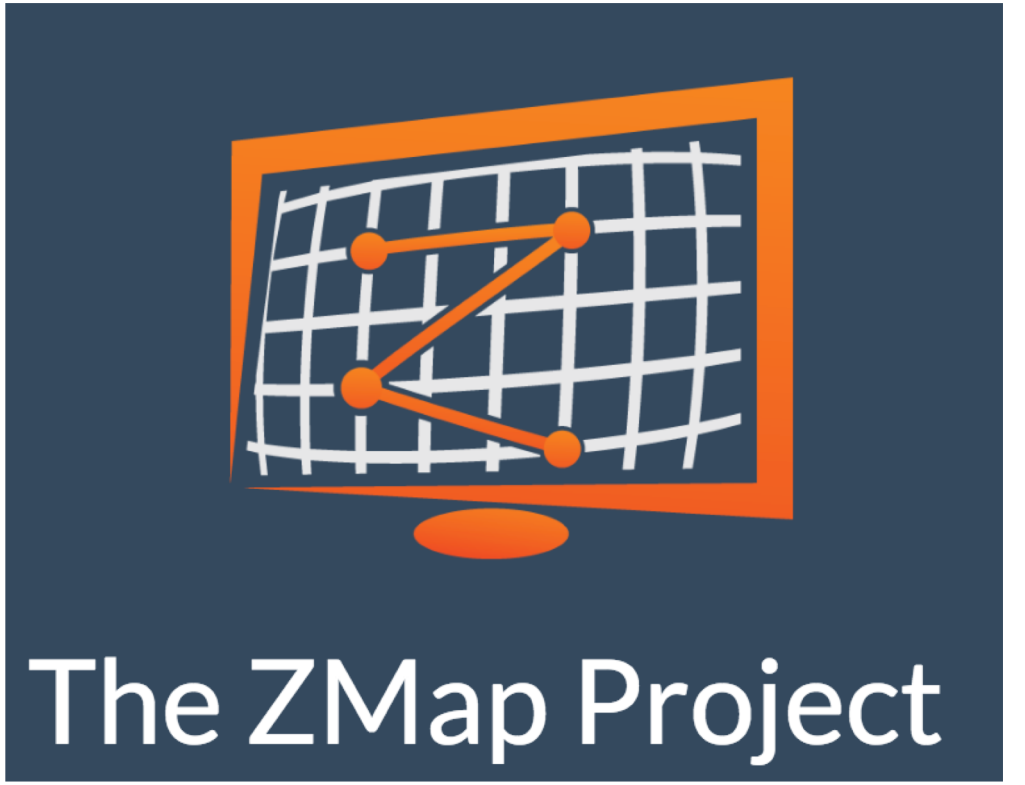
- **Target Selection**: Measurement targets should be chosen automatically and dynamically.
- **Scaling**: Each component can be horizontally scaled if more scaling is required
- **Microservices**: Each component must be independent, no tight coupling allowed (except ELK stack). communication should 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/
  - 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

```
1  from hubble.collect.zmap import Zmap
2
3  SKY_PREFIX = '5.64.0.0/13'
4
5  zmap = Zmap()  # Create a Zmap instance
6  # return max 50 Hosts from SKY_PREFIX, scanning bandwidth max is 1Mbps
7  hosts = zmap.icmp_scan('5.64.0.0/13', max_results=50, bandwidth=1000)
8
9  print(hosts)
10 ['5.64.118.25',
11  '5.64.154.81',
12  '5.64.160.228',
13  '5.64.166.63',
14  ...
15  '5.71.233.19',
16  '5.71.236.232',
17  '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 the Internet in parallel (at a specified packets-per-second rate) so that bulk data can be collected in a timely fashion.

Components – Probers

```
1  from hubble.probers.scamper import Scamper
2
3  # provider is a custom tag appened to each scan
4  prober_a = Scamper(provider='transit_provider_a')
5
6  # Ping all in `hosts` list 10 times with 3 seconds interval,
7  # do not summarize and return as dict, (other option pandas df)
8  result = prober_a.get_measurement_icmp(host_list=hosts,
9                                         interval=3,
10                                        probe_count=10,
11                                        only_stats=False)
12
```

```
{'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}}],
  'start': {'sec': 1523823546, 'usec': 340833},
  'statistics': {'avg': 13.316,
   'loss': 0,
   'max': 17.446,
   'min': 12.494,
   '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)
```

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

Components – Enrichers

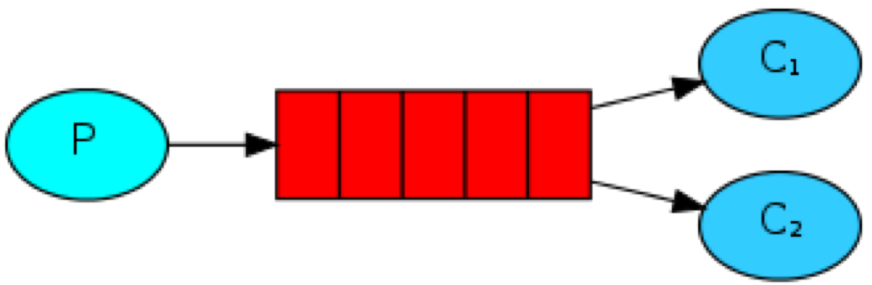
- Measurement data is enriched with BGP and GeoLocation metadata so more insights can be inferred.
- ASN information appended to each scan using pyasn
  - **pyasn** is a Python extension module that enables very fast IP address to Autonomous System Number lookups developed by Economics of Cybersecurity research group at Delft University of Technology
- Elastic's Logstash module appends GeoLocation tag for each scan before sending to EleasticSearch for indexing.

Components – BigData Compute, Storage and Visualization

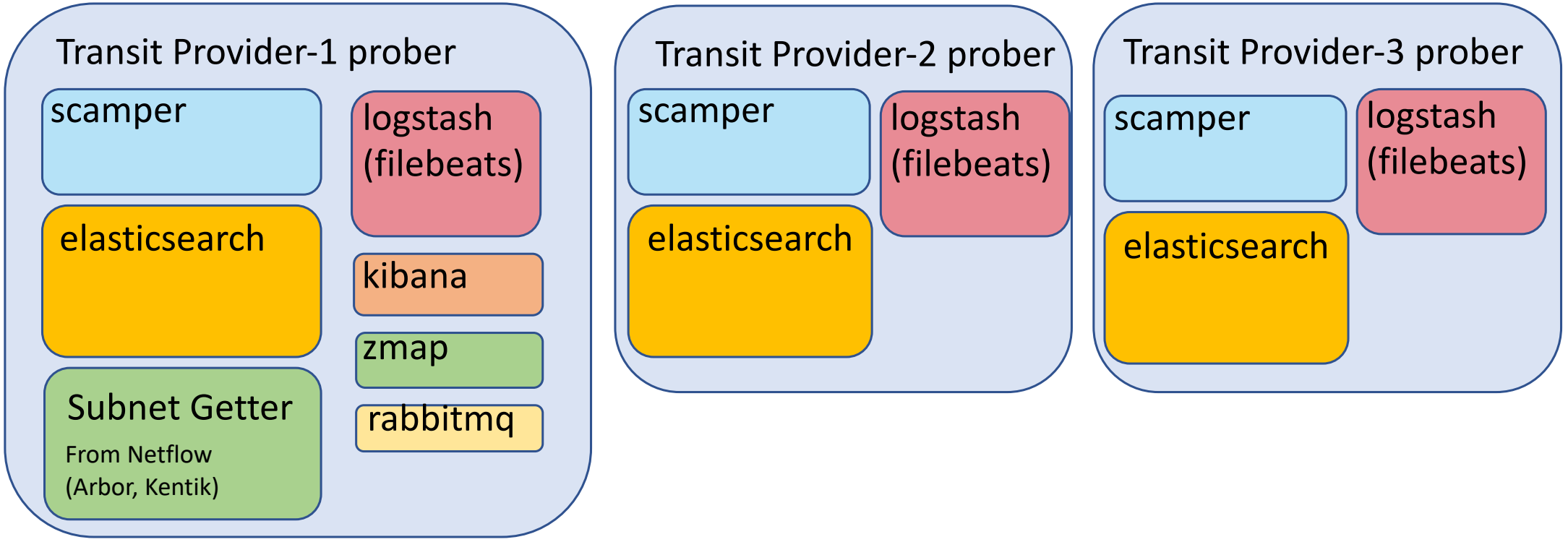
- Measurement data saved to ElasticSearch NoSQL DB.
  - Elasticsearch is a distributed, RESTful search and analytics engine and horizontally scalable.
  - Elasticsearch lets us perform and combine many types of searches; structured, unstructured, geo and metric.
- Kibana is used as visualization and dashboarding platform.
  - Kibana is used for visualizing Elasticsearch data and navigate the Elastic Stack

Components – Message Bus

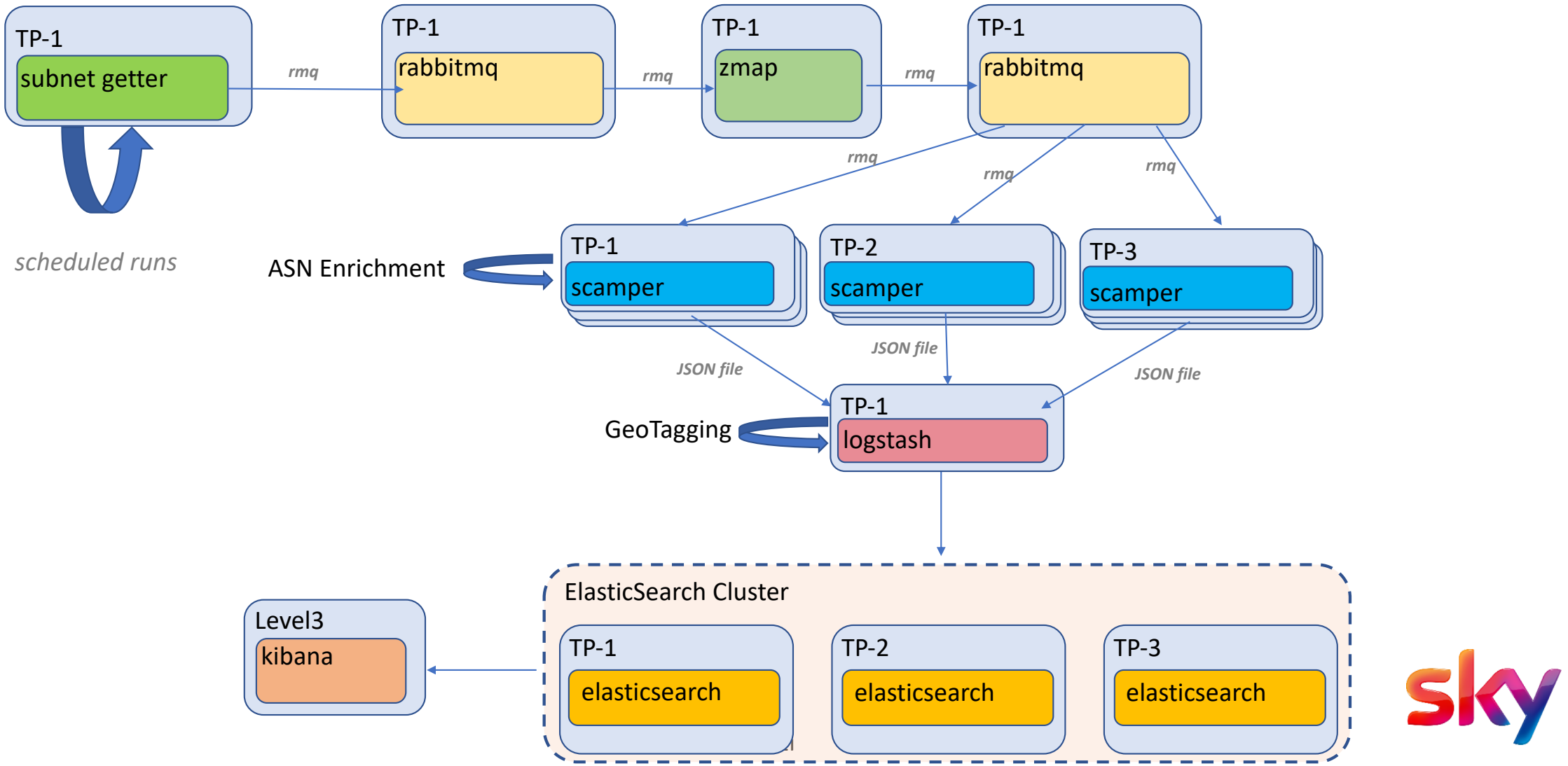
- Data moved between instances of components should be done by a message bus.
- Hubble library has support for rabbitmq, this allows each component to scale independently.
  - For example we may need 10 probers to work in parallel, and each prober subscribes to work queue from Zmap. Message bus is responsible for allocating jobs to each prober instance, allowing parallelism and load sharing.



Software Components

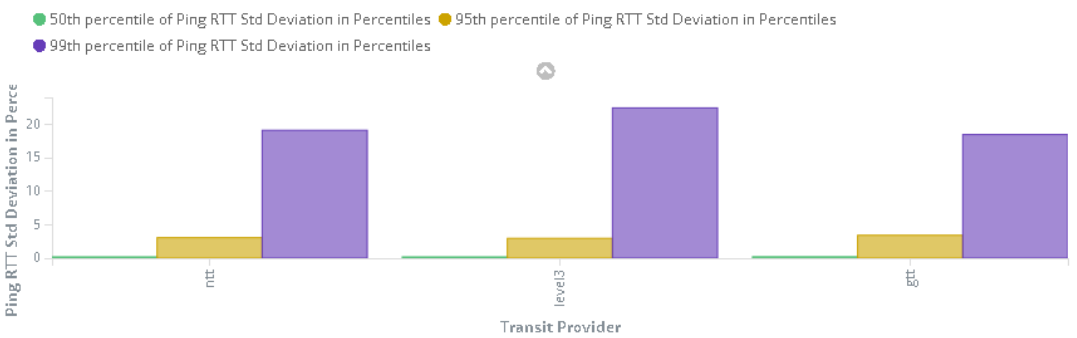


Components Interaction



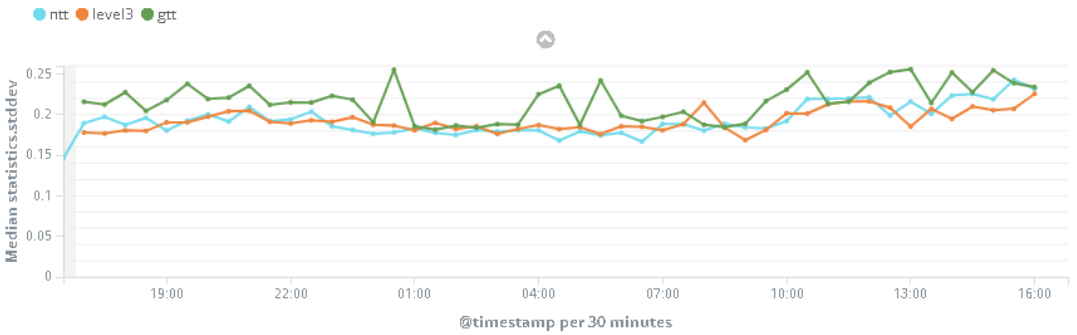
Dashboard: General View

RTT SD vs Provider

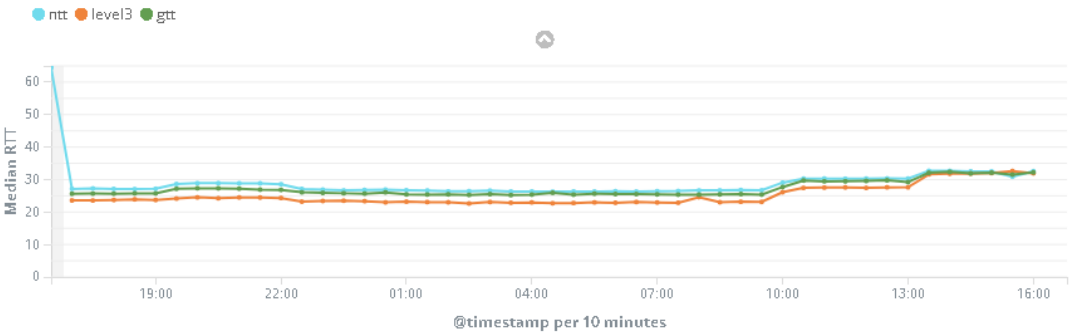


Loss vs Country by Provider

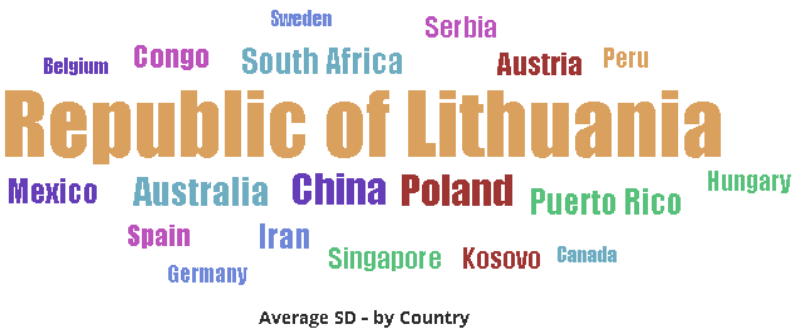
Median STD by Time



Median RTT by Time



SD by Country



Median SD

0.207

50th percentile of Standard Deviation of RTT

Median RTT

28.835

50th percentile of RTT

Mean Packet loss

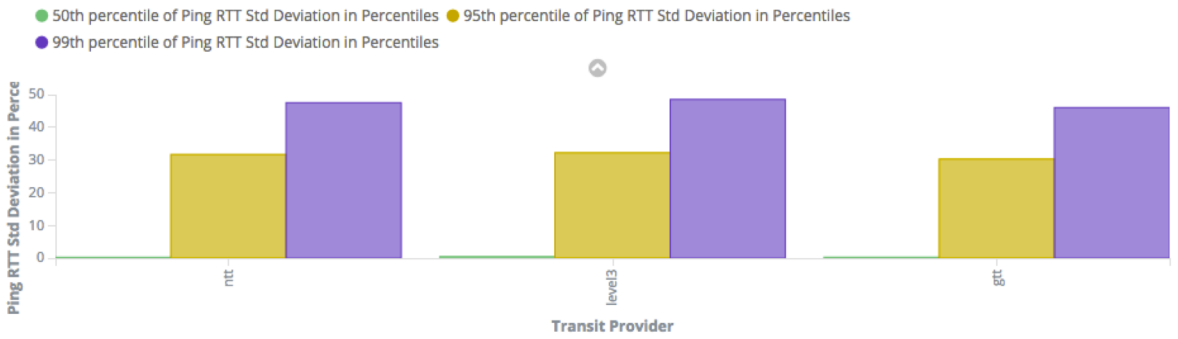
0.027

Average Packet Loss

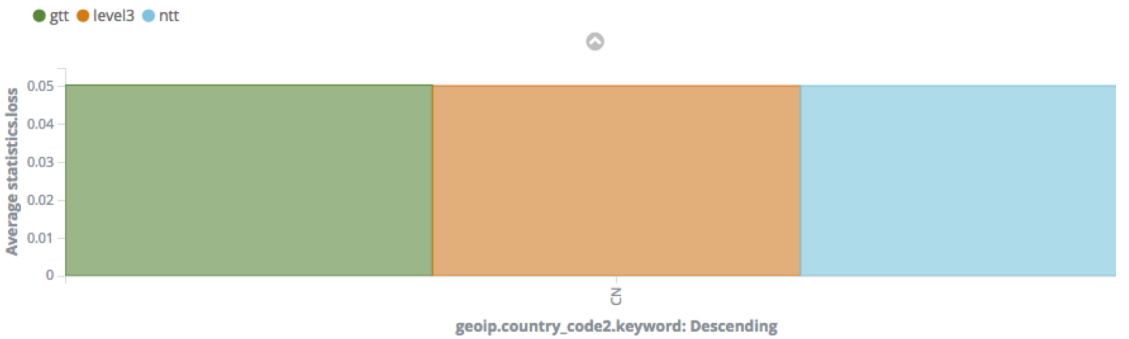
Dashboard: Filter by Country

geoip.country_code2: "CN"

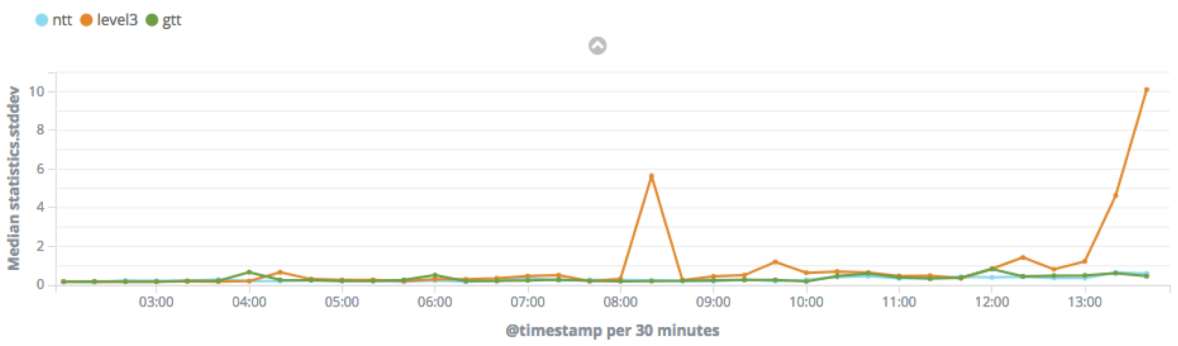
RTT SD vs Provider



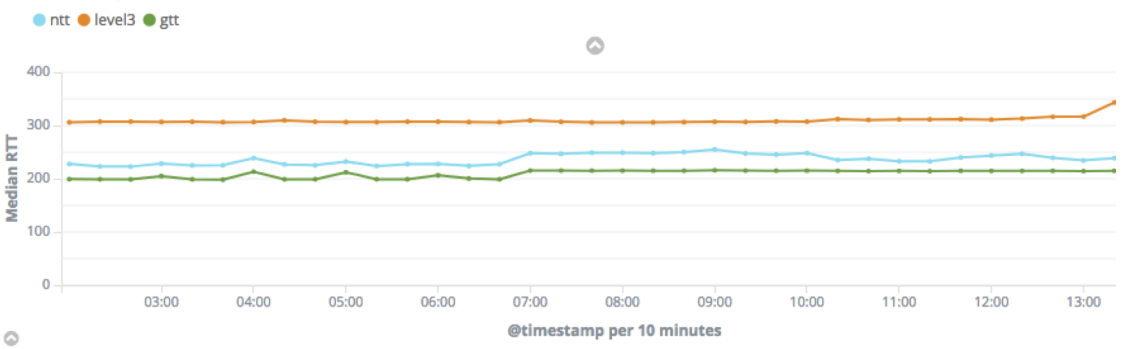
Loss vs Country by Provider



Median STD by Time



Median RTT by Time



SD by Country

Median SD

Median RTT

Mean Packet loss

China

0.338

50th percentile of Standard Deviation of RTT

236.619

50th percentile of RTT

0.05

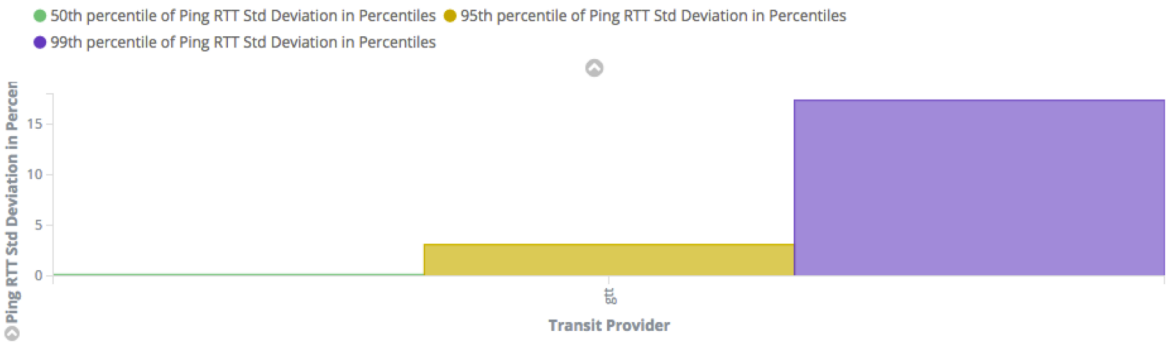
Average Packet Loss

Average SD - by Country

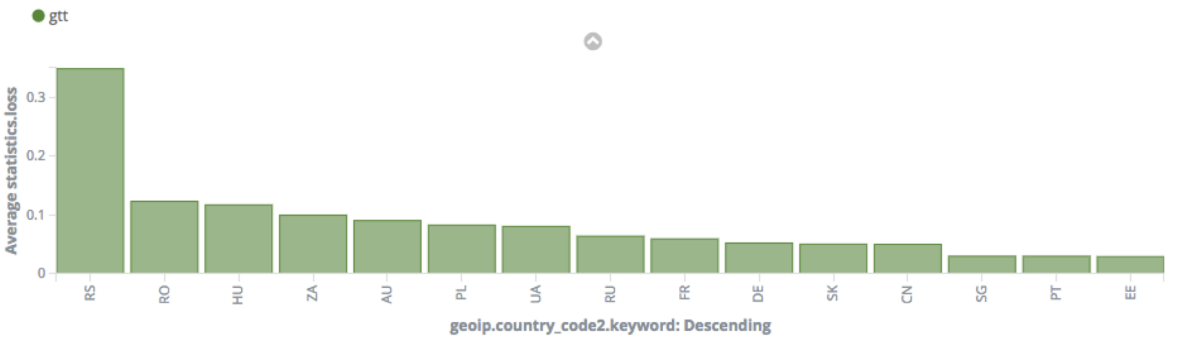
Dashboard: Filter by Transit Provider

_type: gtt

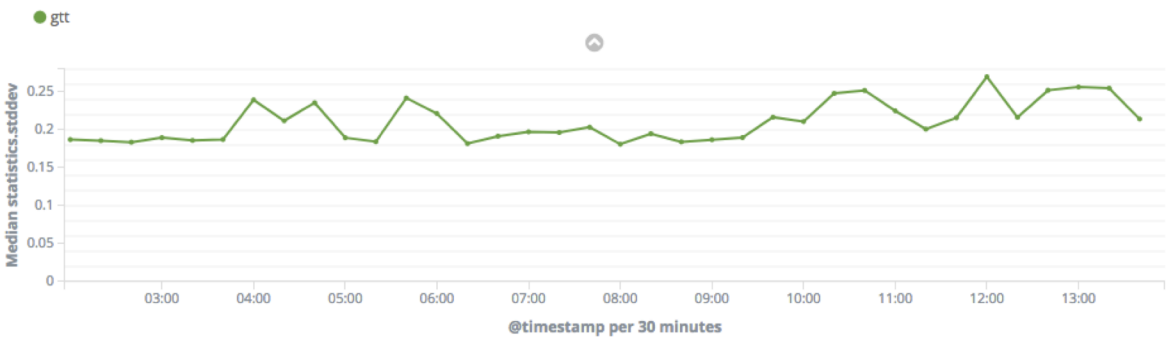
RTT SD vs Provider



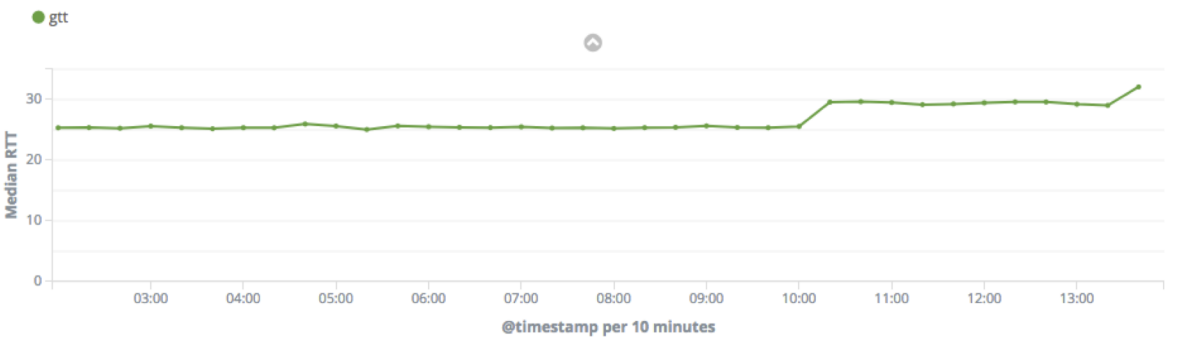
Loss vs Country by Provider



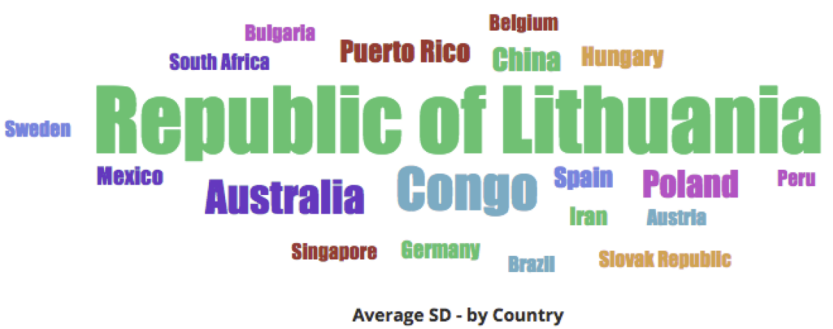
Median STD by Time



Median RTT by Time



SD by Country



Median SD

0.212

50th percentile of Standard Deviation of RTT

Median RTT

27.182

50th percentile of RTT

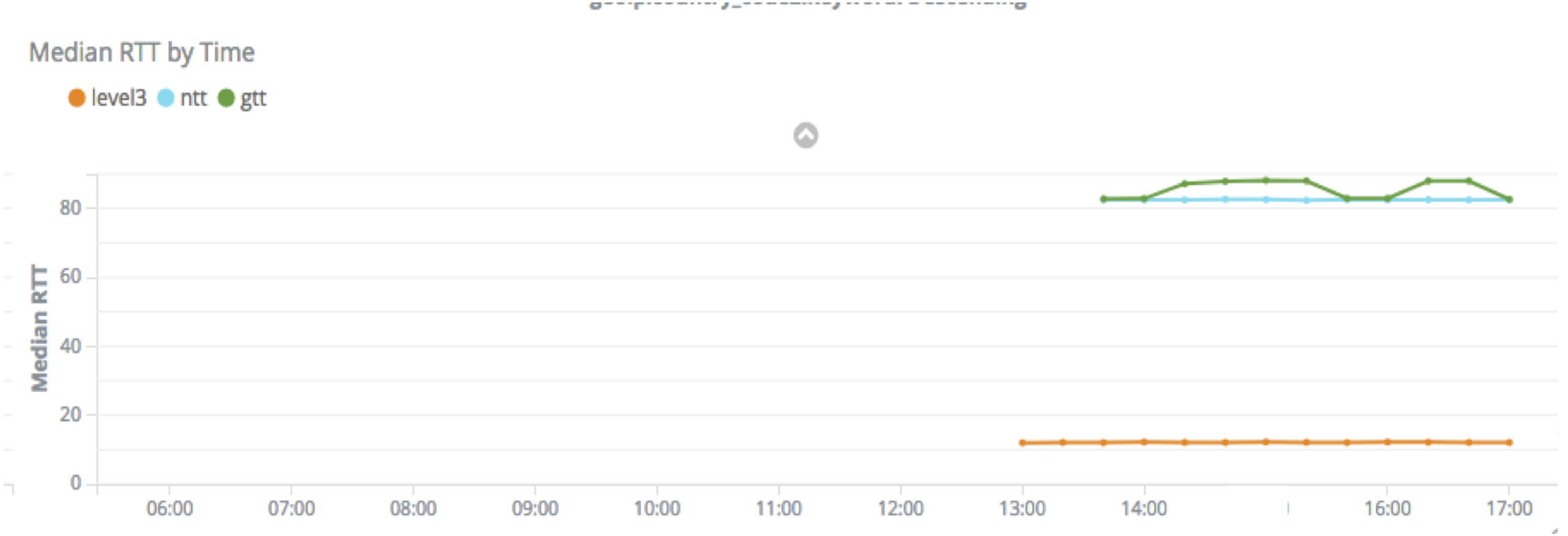
Mean Packet loss

0.029

Average Packet Loss

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.



Limelight Case – GTT

- Digging deeper shows that GTT is connected to Limelight through their PNI in New York

```
traceroute to 178.79.243.158 (178.79.243.158), 30 hops max, 60 byte packets
 1  b0fff001.bb.sky.com (176.255.240.1)  1.589 ms  4.379 ms  5.039 ms
 2  172.16.5.7 (172.16.5.7)  11.327 ms  11.757 ms  12.204 ms
 3  b0ffa222.bb.sky.com (176.255.162.34)  5.493 ms  9.979 ms  10.538 ms
 4  b0ffa225.bb.sky.com (176.255.162.37)  6.355 ms  7.021 ms  9.276 ms
 5  b0ffa226.bb.sky.com (176.255.162.38)  7.540 ms  8.655 ms  8.064 ms
 6  ae1.cr0-lon8.ip4.gtt.net (77.67.65.173)  12.293 ms  2.622 ms  3.224 ms
 7  xe-2-0-0.cr8-nyc3.ip4.gtt.net (89.149.183.38)  70.258 ms  71.482 ms  70.781 ms
 8  ip4.gtt.net (69.174.2.178)  68.866 ms  69.531 ms  71.948 ms
 9  lag28.fr6.lon.llnw.net (68.142.88.61)  72.418 ms  72.897 ms  70.835 ms
10  lag27.fr4.fra1.llnw.net (68.142.88.90)  86.191 ms  86.606 ms  87.078 ms
11  vl2023.dr01.fra1.llnw.net (178.79.240.22)  85.212 ms  85.682 ms  85.145 ms
```

Limelight Case - NTT

- NTT does not have direct peering with Limelight so it connects through GTT

```
traceroute to 178.79.243.158 (178.79.243.158), 30 hops max, 60 byte packets
 1  b0fff101.bb.sky.com (176.255.241.1)  1.724 ms  2.174 ms  2.780 ms
 2  172.16.5.7 (172.16.5.7)  11.043 ms  11.443 ms  11.826 ms
 3  b0ffa222.bb.sky.com (176.255.162.34)  3.159 ms  3.547 ms  4.408 ms
 4  b0ffa225.bb.sky.com (176.255.162.37)  5.390 ms  6.401 ms  7.530 ms
 5  b0ffa226.bb.sky.com (176.255.162.38)  6.980 ms  8.823 ms  8.101 ms
 6  ae-1.r00.londen01.uk.bb.gin.ntt.net (83.231.199.161)  10.146 ms  3.925 ms  4.472 ms
 7  ae-8.r25.londen12.uk.bb.gin.ntt.net (129.250.3.2)  3.902 ms  4.832 ms  3.029 ms
 8  ae-22.r00.londen10.uk.bb.gin.ntt.net (129.250.4.40)  3.329 ms  4.671 ms  3.600 ms
 9  ae8.cr10-lon1.ip4.gtt.net (141.136.96.181)  6.006 ms  6.179 ms  5.773 ms
10  et-9-1-0.cr9-nyc3.ip4.gtt.net (89.149.138.230)  73.775 ms et-5-3-0.cr9-nyc3.ip4.gtt.net (89.149.187.189
11  ip4.gtt.net (69.174.2.206)  73.462 ms  73.190 ms  72.894 ms
12  lag29.fr5.lon.llnw.net (68.142.88.59)  80.367 ms ve5.fr4.lga.llnw.net (69.28.172.206)  73.754 ms lag29.
13  lag28.fr6.lon.llnw.net (68.142.88.61)  80.513 ms  80.709 ms tge1-5.fr6.lon.llnw.net (178.79.195.62)  80
14  lag27.fr4.fra1.llnw.net (68.142.88.90)  118.298 ms  118.491 ms  117.084 ms
15  vl2024.dr02.fra1.llnw.net (178.79.240.26)  96.884 ms vl2023.dr01.fra1.llnw.net (178.79.240.22)  96.753
```

Limelight Case – Level 3

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

```
traceroute to 178.79.243.158 (178.79.243.158), 30 hops max, 60 byte packets
 1  b0fff201.bb.sky.com (176.255.242.1)  0.784 ms  0.981 ms  1.098 ms
 2  172.16.5.7 (172.16.5.7)  3.516 ms  3.720 ms  3.893 ms
 3  b0ffa222.bb.sky.com (176.255.162.34)  1.248 ms  1.441 ms  1.615 ms
 4  b0ffa225.bb.sky.com (176.255.162.37)  2.876 ms  3.085 ms  3.921 ms
 5  b0ffa226.bb.sky.com (176.255.162.38)  1.914 ms  2.138 ms  2.320 ms
 6  lag-115.ear2.London2.Level3.net (195.50.116.125)  1779.522 ms  1751.281 ms  1669.027 ms
 7  Cogent-level3-100G.London2.Level3.net (4.68.72.186)  5.544 ms be3356.ccr21.lon01.atlas.cogentco.com (130.117.14.133)  3.503 ms
 8  be2208.rcr21.b015589-1.lon01.atlas.cogentco.com (154.54.37.66)  7.809 ms be2950.rcr21.b023101-0.lon01.atlas.cogentco.com (130.1
 9  149.6.147.150 (149.6.147.150)  3.822 ms 149.6.147.202 (149.6.147.202)  4.785 ms limelight.demarc.cogentco.com (149.14.147.210)
10  178.79.248.12 (178.79.248.12)  6.138 ms lag9.fr4.cdg1.llnw.net (68.142.88.111)  12.924 ms tge1-5.fr6.lon.llnw.net (178.79.195.6
11  lag1.fr3.cdg1.llnw.net (185.178.52.12)  13.212 ms lag12.fr3.ams.llnw.net (68.142.88.66)  12.458 ms lag27.fr4.fra1.llnw.net (68.
12  vl2023.dr01.fra1.llnw.net (178.79.240.22)  17.763 ms * *
13  vl2013.dr01.fra1.llnw.net (178.79.240.14)  13.201 ms  16.271 ms *
```

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.
- 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.