How does UBER check the health of its Network Infrastructure every 10 sec?

Vasileios Lakafosis, Ph.D.
Senior Software Engineer
UBER

github.com/uber/Arachne
Agenda

- Why?
- The UBER Scale
- Monitoring → Auto-detection
- Arachne
  - The Agents (data plane)
  - The Orchestrator (control plane)
- Measurement Examples and Business Impact
Why?
Our Hyper-scale Growth
Moving people, products, and business

560+
cities Uber operates in, in 74+ countries.

2Bth
trip completed in June 2016. 1Bth trip in Dec. 2015.

5M+
trips that happen each day, on average.

40M+
monthly active riders.

1.5M+
active drivers globally. More than 600K in the US.

2K+
production microservices running on our infrastructure.
The UBER Network Infrastructure
TCP in the Data Center

TCP accounts for **99.9% of traffic** in the Data Center

TCP suffers from **packet drops** [SIGCOMM 2009+2010]

- Considers drops as sign of congestion
- Minimum retransmit time-out is 200 msec [RFC 1122 from late 1980s]
- Builds up queues, adds latency, and harms throughput

The two network metrics that have the most impact on the performance of TCP:

1. path latency and
2. packet loss ratio
How do we check the health of the **Network Infrastructure** (DCs, PoPs, and Cloud) every **10 sec**?
With always-on, active traffic monitoring!
Why develop our own solution in-house?

Limitations in tools provided by the original manufacturers:

- may suffer from software glitches
- may not provide the full picture + tools differ between them
- may not provide a flexible way to extract the metrics
Why develop our own solution in-house?

By designing and creating our own system ➔

- we **tailor** it for our own needs ➔
  - always-on, active system on top of any hardware
  - end-to-end functional testing **every 10 sec** using TCP
  - covering the entire network infrastructure (across Data Centers, PoPs, and Cloud)
  - probing external 3\textsuperscript{rd}-party services
  - we **add features** and **fix bugs**, as fast as we, ourselves, can develop!
Why develop our own solution in-house?

We own the system ➔ we open source it

- strengthen its quality and
- enrich its feature set

Hopefully, useful to other people and organizations!
intra-DC, inter-DC, Cloud-to-DC, and DC-to-External Services metrics reported by Arachne

Directly measured
- Reachability
- Round-trip and one-way latency
- Packet drops (silent or bursty) and black holes

Indirectly measured
- Jitter (average of the deviation from the network mean latency)
- Firewall or PMTU issues related probably to network config changes
- Whether Network-level SLAs are met
ECMP in the Data Center — Inter-Cluster
ECMP in the Data Center — Inter-Cluster
ECMP in the Data Center — Inter-Cluster
ECMP in the Data Center — Inter-Cluster
ECMP in the Data Center — Inter-Cluster
ECMP in the Data Center — Inter-Cluster
ECMP in the Data Center — Inter-Cluster
ECMP in the Data Center — Inter-Cluster
The Agents

The data plane of Arachne
A single ‘echo’

Benefits for not establishing the TCP connection

- Closest-to-real measurement (doesn’t measure lag of establishing the session in overall latency)
- Minimum computational and traffic overhead on both sides

**Half-Open TCP connection between two agents**

**Unestablished TCP connection between an agent and a 3rd-party services**
Echo Parameters

Testing **ECMP** (Equal Cost Multi-path)
- Apply different five-tuple hashes by changing the ephemeral source port

Testing **QoS** (Quality of Service)
- Classes *BE, Bulk, NC, Tier 0, Tier 1, and Tier 2* by applying 11 different DSCP values embedded in the ToS field
Data Collection, Visualization & Alerting

UBER home-grown time-series database & query language

UBER home-grown alerting systems

Arachne agent

filebeat

logstash

elasticsearch

kafka

grafana

kibana
The Orchestrator

The **control plane** of Arachne
The Orchestrator Architecture

1st Generation: in the DC

2nd Generation: on the Cloud
Agent’s Target Configuration File from Orchestrator

```
{
    "local": {
        "region": "us-west",
        "src_address": "",
        "interface_name": "eth0",
        "target_tcp_port": 44111,
        "timeout_msec": "200ms",
        "base_src_tcp_port": 31000,
        "num_src_tcp_ports": 64,
        "batch_interval_sec": "10s",
        "qos": "enabled",
        "resolve_dns": "enabled",
        "dns_servers_alternate": "8.8.8.8",
        "poll_orchestrator_interval_success": "2h",
        "poll_orchestrator_interval_failure": "15m",
    },

    "internal": [
        {"ip": "10.10.10.1"},
        {"host_name": "hadoop375.internal.servers"}
        ...
    ],

    "external": [
        {"host_name": "payments.externalservice.com"},
        {"host_name": "messaging.externalservice.com"}
        ...
    ]
}
```

<table>
<thead>
<tr>
<th>HTTP/REST API response code from Orchestrator</th>
<th>Agent’s State</th>
<th>Target configuration file downloaded?</th>
<th>Retry interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 — OK</td>
<td>active</td>
<td>Yes</td>
<td>2 h</td>
</tr>
<tr>
<td>404 — not found</td>
<td>idle</td>
<td>No</td>
<td>2 h</td>
</tr>
<tr>
<td>500 — internal server error</td>
<td>prior state</td>
<td>No</td>
<td>15 min</td>
</tr>
<tr>
<td>400 — bad request</td>
<td>down → removed</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>down → removed</td>
<td>No</td>
<td>&gt; 6 h</td>
</tr>
</tbody>
</table>
Target List Generation Requirements

When configuring the target list (set of target agents to be echoed), the following are taken into account:

- **Per-rack granularity**: There must be two agents *active* in every rack.
- It is neither feasible, nor necessary for a single server to echo every other server of the entire Uber infrastructure.
- The other agents being echoed must be *distributed evenly throughout the global infrastructure*.
- If an agent is known to be *offline*, all other agents should not attempt any tests against it.
## Even Echo Target Distribution

| Ring #0  | agents within the same pod |
| Ring #1  | agents in other pods of the same cluster |
| Ring #2  | agents in other clusters of the same DC |
| Ring #3  | agents in other DCs, PoPs, and Cloud regions |
| Ring #4  | external services owned by 3rd parties |
### Even Echo Target Distribution

**Rings (sets) of targets echoed by each active agent in every echoing interval**

<table>
<thead>
<tr>
<th>Ring #0</th>
<th>agents within the same pod</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ring #1</td>
<td>agents in other pods of the same cluster</td>
</tr>
<tr>
<td>Ring #2</td>
<td>agents in other clusters of the same DC</td>
</tr>
<tr>
<td>Ring #3</td>
<td>agents in other DCs, PoPs, and Cloud regions</td>
</tr>
<tr>
<td>Ring #4</td>
<td>external services owned by 3rd parties</td>
</tr>
</tbody>
</table>
## Even Echo Target Distribution

<table>
<thead>
<tr>
<th>Ring</th>
<th>Targets Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ring #0</td>
<td>agents within the same pod</td>
</tr>
<tr>
<td>Ring #1</td>
<td>agents in other pods of the same cluster</td>
</tr>
<tr>
<td>Ring #2</td>
<td>agents in other clusters of the same DC</td>
</tr>
<tr>
<td>Ring #3</td>
<td>agents in other DCs, PoPs, and Cloud regions</td>
</tr>
<tr>
<td>Ring #4</td>
<td>external services owned by 3rd parties</td>
</tr>
</tbody>
</table>
Even Echo Target Distribution

Rings (sets) of targets echoed by each active agent in every echoing interval:

- **Ring #0**: agents within the same pod
- **Ring #1**: agents in other pods of the same cluster
- **Ring #2**: agents in other clusters of the same DC
- **Ring #3**: agents in other DCs, PoPs, and Cloud regions
- **Ring #4**: external services owned by 3rd parties
**Even Echo Target Distribution**

<table>
<thead>
<tr>
<th>Ring</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ring #0</td>
<td>agents within the same pod</td>
</tr>
<tr>
<td>Ring #1</td>
<td>agents in other pods of the same cluster</td>
</tr>
<tr>
<td>Ring #2</td>
<td>agents in other clusters of the same DC</td>
</tr>
<tr>
<td>Ring #3</td>
<td>agents in other DCs, PoPs, and Cloud regions</td>
</tr>
<tr>
<td>Ring #4</td>
<td>external services owned by 3rd parties</td>
</tr>
</tbody>
</table>
**Even Echo Target Distribution**

<table>
<thead>
<tr>
<th>Ring</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ring #0</td>
<td>agents within the same pod</td>
</tr>
<tr>
<td>Ring #1</td>
<td>agents in other pods of the same cluster</td>
</tr>
<tr>
<td>Ring #2</td>
<td>agents in other clusters of the same DC</td>
</tr>
<tr>
<td>Ring #3</td>
<td>agents in other DCs, PoPs, and Cloud regions</td>
</tr>
<tr>
<td>Ring #4</td>
<td>external services owned by 3rd parties</td>
</tr>
</tbody>
</table>
Measurement Examples
ECMP Average Round-Trip-Time Latency

6 different sets of latencies being discerned among 16 five-tuple ECMP hashes
Dropped & Timed-out Packets within echo interval

Timed-out response received (>200ms)

Dropped packets (never received a response)
Business Impact
Transit Provider Circuit Outage during a busy Saturday evening

Fast detection means, almost zero customer impact.
Average Latency (RTT) as seen from 2 DCs against each other
ECMP Average Latency (RTT)
Cloud provider → DC

Almost 40% improvement
Average Latency (RTT) to external 3rd-party services

2 short-lived instances of degradation

1 slowly-building long-lived instance of degradation
Thank you

Joint work with Henri Devieux.

Download Arachne at [github.com/uber/arachne](https://github.com/uber/arachne)

More open-source projects at [UBER Engineering](https://eng.uber.com)