Scalable Routing Information Base for the Global Data Plane (GDP)

Li XiangJun (Jason)

University of California Berkeley

Abstract

The Global Data Plane (GDP) provides a flat address space routing protocol to endpoints (e.g. DataCapsules) identified by 256-bit hash names, rather than IP addresses. DataCapsules are served over the network on DataCapsule servers, which "Advertise" the logs that they support. These advertisements are signed delegations from the DataCapsule owner stating that the owner of the DataCapsule server is allowed to advertise.

A core component in the realization of the location independent naming of clients and DataCapsules in a GDP network is routing. The current GDP routing infrastructure consists of a two-layer communication scheme. A lower, routing overlay layer, connecting GDP clients: applications, services, and DataCapsule servers; and an upper location resolution layer which tracks the location of DataCapsule endpoints by mapping 256-bit hashed names to locations and other nameable resources.

Our work aims to deliver a sketch of an architecture and also an implementation of the location resolution layer that is scalable, secure and supports multiple trust domains.

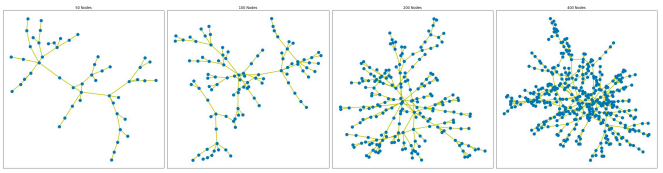
Architecture

- We do not assume any point-to-point connectivity at the switch layer. Each switch is connected with other switches on a L2 layer. They connect through dTLS with each other.
- Every switch needs to advertise their neighbors to the RIB which keeps track of the top-level topology.
- RIB acts as the oracle and computes routes so that switches know where to send the packet to for the next hop. RIB also caches the computed route so that when the subsequent switches ask for routes, routes do not have to be recomputed.
- The RIB maintains two databases.
    - The first database maintains a key value store for the mapping from log hash to switch id. Note that one log hash could be mapped to multiple switch ids as one log could simultaneously be served by multiple log ids. We anticipate trillions of logs in the system so the key-value look-up has to be efficient. Here we used distributed key value service Chord. Chord is very scalable because each node in the system does not have to know all the other nodes. It just needs to maintain connections with a subset of the nodes for lookup to happen efficiently.
    - The second database is to maintain the network topology and compute routes. Route computation is reliant on the underlying data and we can run many instances of it. The topology data is easy to fit in memory for one million nodes. If we assume that each switch is on average have 5 outgoing connects with 5 other top-level switches, the total memory required to store this topology information would be (10 ** 6) * 5 * 256 / 8 / 10 ** 9 < 2 GB. Graph computation on such network graph has been proved to be efficient too. We were able to get similar results in our experiment.
- A top-level switch maintains the following:
    - Log entries for those this switch is a host for and how to reach the log servers with in a trust domain.
    - Cached routes to reach some log hashes so that RIB does not need to be consulted again.
    - A list of L2 connected peers in the system.

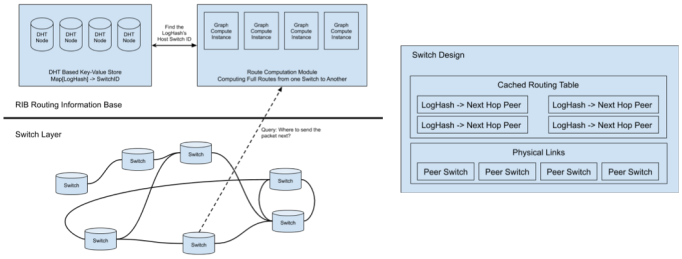
Simulation

Out simulation is based on generated graphs as we are unable to acquire real-world data sets that have as many nodes as we want there to be and is also consistent in terms of topology for different number of nodes.

Our graph generation algorithm connects a newly joined node randomly with a node that is already in the system with uniform probability, there by generating a density-constraint connected graph. The generated topology is in fact similar to the real topology as suggested by some papers. We have included how the network would look like for 50, 100, 200, 400 nodes below.



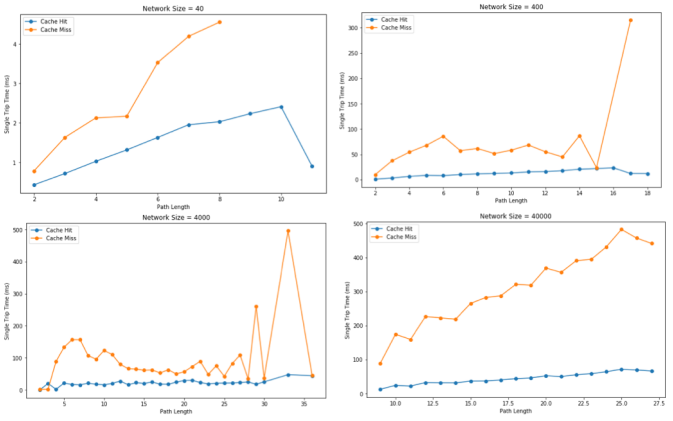
Generated Network Topology



System Architecture for RIB and switches

Results and Conclusions

In order to investigate if such an architecture is a step towards a scalable infrastructure, we have set up to 40,000 virtual switches and billions of logs using a few xlarge instances on AWS and measured the single packet trip time of such systems when deployed. Each switch is given a forwarding delay of a normal distribution of around 3 milliseconds. We instruct the system to send 1 thousand packets between tow randomly chosen nodes and collected the results. We then instruct the system to send 1 thousand packets to the same set of source-destination again to measure the trip time if there is route cache.



- When there is routing entry cache, the forwarding time taken is a lot smaller than when there is not.
- While graph computation time increases with the network size, the single trip time for 40,000 nodes looks acceptable and can even be further improved. (Note that the above results are generated when there is no cache in the RIB which means that each time a switch asks for routes, the RIB has to recompute.)
- Vanilla Dijkstra might not be able to scale up to 1 million nodes for this purpose because for routes that are hard to compute, it would take a long time as evidenced by the spikes in the graph. Further research can look into how to add heuristics to this computation. For example, maybe the geo-location of the switches can be used as a guide.

References

Guor-Huar Lu, Sourabh Jain, Shanzhen Chen, and Zhi-Li Zhang. 2008. Virtual id routing: a scalable routing framework with support for mobility and routing efficiency. In Proceedings of the 3rd international workshop on Mobility in the evolving internet architecture (MobiArch '08). ACM, New York, NY, USA, 79-84. DOI: https://doi.org/10.1145/1403007.1403025

Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. In Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '01). ACM, New York, NY, USA, 149-160. DOI=http://dx.doi.org/10.1145/383059.383071

Feng Hong, Minglu Li, Jiadi Yu and Yi Wang, "PChord: improvement on chord to achieve better routing efficiency by exploiting proximity," 25th IEEE International Conference on Distributed Computing Systems Workshops, Columbus, OH, 2005, pp. 806-811.

Prasanna Ganesan, Krishna Gummadi and H. Garcia-Molina, "Canon in G major: designing DHTs with hierarchical structure," 24th International Conference on Distributed Computing Systems, 2004. Proceedings., Tokyo, Japan, 2004, pp. 263-272.

Aviram, N. and Shavitt, Y. (2015). Optimizing dijkstra for real-world performance. Networking and Internet Architecture - arXiv preprint arXiv.