Flow Processing Platform discovery

Octavian Rinciog
Computer Science Department
University Politehnica of Bucharest
octavian.rinciog@cs.pub.ro

Costin Raiciu
Computer Science Department
University Politehnica of Bucharest
costin.raiciu@cs.pub.ro

1. INTRODUCTION

Flow processing is a manipulation of packets inside the network where the service given to a set packets is differentiated based on their implicit membership of a labeled flow. Flow processing is already taking place in the Internet at different vantage points today, with boxes like firewalls, NATs, performance enhancing proxies, application-level gateways, etc., looking at layer four and above to do their job. These boxes quite prevalent: as we speak, more than a third of the paths probed in a recent study show signs of L4+ middlebox behavior [2].

Instead of fighting this trend, we choose to embrace it. The CHANGE project [1] wants to create a new Internet architecture where flow processing is not only permitted, but encouraged. In the CHANGE vision, endpoints can request and instantiate in-network processing from third parties deploying flow-processing functionality. The resulting network is both more flexible and easier to reason about than the Internet we have today.

For flow processing to be viable, a number of mechanisms are needed, including authentication of traffic owners, payment, discovery of platforms, and so forth. In this poster we focus on the platform discovery platform: if an entity wants its flow processed, how can it find a suitable processing platforms? The position of the platform influences the communication delay between the source and the destination. In order to make flow processing efficient, it is important that the end-to-end delay stays low; this is currently the single most important reason affecting user behavior.

A good platform discovery algorithm must find a number *k* of processing platforms closest to the requesting entity. Out of this set, the entity can choose the one that meets other arbitrary requirements, for instance network and processing capacity. Another important requirement is that ISP's routing policies should be obeyed, else flow processing will never be deployed.

2. EXISTING SOLUTIONS

The problem we want to tackle is similar to finding the best mirror(s) for a site. In the literature, there are two classic solutions to this problem:

- Using IP Anycast [3]
- Using DNS plus geolocation [6]

IP Anycast is used by root DNS servers in order to route root DNS requests to the server closest to the client. For our needs, each processing platform can be configured with the same global anycast IP address within a given prefix. Each processing platform then acts as a BGP stub and announces this global prefix such that it is distributed on the global Internet scale.

This solution is very efficient for locating nearby platforms: a single message suffices to locate the closest platforms. Using one anycast IP only finds the single closest platform, which is not flexible enough for our needs.

Another solution based on DNS and geolocation is used by well-known CDNs, like Akamai or Amazon. The DNS server holds a database of all platform locations. When a request arrives, the platform uses geolocation to map the client into a geographical area. Then, it chooses the *k* platform(s) that are geographically closest to the client. This solution is based on the assumption that geographical proximity is correlated with low network delay, which is generally true. However, this solution is not 100% accurate: it breaks down when geolocation is not accurate, as is the case for provider-independent addresses.

Another disadvantage of this solution is the need to have a global database of platform locations. This is entirely feasible for CDNs like Akamai, but may be infeasible in the federated model of the Internet. Internet Service Providers may be reluctant to give away the exact positioning of the platforms, as it may be regarded as confidential information.

Finding the *k* closest platforms to requester can be done using network coordinates[5] or systems like Meridian[7], but these systems ends up being inaccurate because of triangle-inequality violations in the Internet, and are not compliant with ISP policies. Further, the discovery latency and the number of messages needed are quite big.

3. OUR PROPOSAL

The only existing solution which preserves routing policies is based on IP Anycast, but the solution only finds the closest platform to the requester. An obvious extension is to have *k* or more anycast addresses, split the platforms into *k* groups and assign one address to all the hosts in the same group. The important question is: how do we assign these addresses such that the platforms found by each host via IP anycast are indeed the *k* closest platforms to it? And if they are not the *k* closest, how do we minimize the average increase in delay?

We want as few IP anycast addresses as possible: using many IP anycast addresses has an overhead as it increases

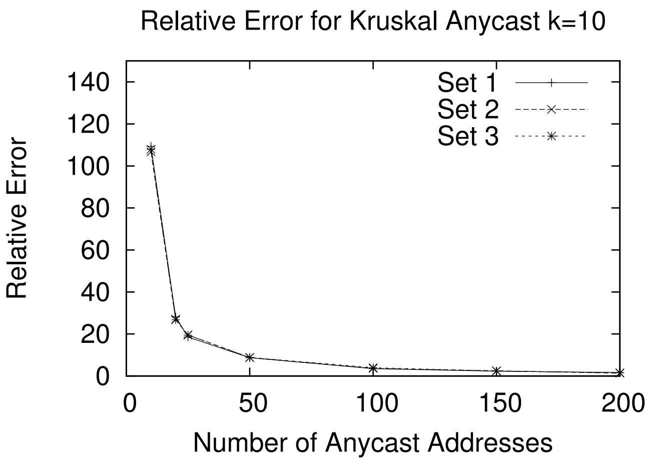

Figure 1: Relative error for $N = 1000$ platforms

the total number of BGP UPDATE messages in the global routing system.

In the following, we present our solution which uses the classical Kruskal minimal spanning tree algorithm as a starting point. The algorithm assigns x anycast addresses to N platforms ($k \leq x \leq N$), where x is a parameter. The idea is to split these N platforms in clusters with maximum x members, each of these members being connected to the other members with the minimum cost edge; this ensures that nearby servers are placed in the same clusters. To allow precise lookups, inside these clusters each platform has a different anycast address.

Our algorithm has two distinct parts. First, split platforms into clusters where each member has a different address. Second, assign addresses to each cluster member.

The first part is done using a modified Kruskal[4] spanning-tree algorithm. From the original algorithm is kept the idea of merging two different clusters if an edge is found a edge with certain properties is found. The properties we require are: a) the edge connects two different clusters, b) the edge has the minimum cost and c) the total number of vertices in the two clusters must be lower than x.

After each cluster is formed, we must assign to each member a different anycast address. The assignment of addresses is made taking the edges in descending order of cost and assign the same minimum possible address to the platforms connected by one edge.

3.1 Algorithm Evaluation

We measured the performance of Kruskal Anycast algorithm on a simulation of 1000 platforms. All platforms are directly connected and the delay between any two platform is chosen randomly between 0 and 1000ms. We run our algorithm using three random topologies, and fixed $k = 10$ and varied x between 10 and 200.

For each value of x anycast addresses, we calculated the mean delay to the closest k platforms computed by our algorithm, and compared these results with the mean delay to the true k closest platforms. We show our results in figure 1.

Algorithm 1 Assign x anycast addresses to N platforms

```
Require: $x \leq n \wedge G =$ complete delay graph
  Sort each edge from G in ascending order
  Put each platform in its own cluster
  $i \leftarrow 0$
  while $\#each\_cluster < x \vee i < N$ do
    $u \leftarrow leftnode_{edge_i}$
    $v \leftarrow rightnode_{edge_i}$
    if $cluster_u \neq cluster_v \wedge \#cluster_u + \#cluster_v \leq x$ then
      Merge $cluster_u$ $cluster_v$
    end if
    $i++$
  end while
  $i \leftarrow N - 1$
  while $i \geq 0$ do
    $u \leftarrow leftnode_{edge_i}$
    $v \leftarrow rightnode_{edge_i}$
    if $cluster_u \neq cluster_v$ then
      $min_{address}$ =find_address($cluster_u, cluster_v$)
      $address_u \leftarrow min_{address}$
      $address_v \leftarrow min_{address}$
    end if
  end while
```

As we can see from graphic, the results aren't influenced by the topology, but are influenced by the number of announced addresses used. As expected, the more addresses we announce the smaller the error. Interestingly, if we announce as little as 20 addresses, the error decreases to 20%; this seems reasonable.

4. CONCLUSIONS

We have presented a solution to the platform discovery problem needed for Internet-wide, flexible flow processing. Our algorithm is a viable first step, but is not directly applicable as it requires centralized knowledge of platform-to-platform delays. We are currently investigating a distributed, online version of the same algorithm.

5. REFERENCES

[1] Change eu fp7 project. www.change-project.eu.
[2] Michio Honda et al. Is it still possible to extend tcp? In *Proc. ACM IMC*, 2011.
[3] Michael J. Freedman, Karthik Lakshminarayanan, and David Mazières. Oasis: anycast for any service. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3*, NSDI'06, pages 10–10, Berkeley, CA, USA, 2006. USENIX Association.
[4] J. B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. In *Proc. of the American Mathematical Society, 7*, 1956.
[5] T. S. Eugene Ng and Hui Zhang. Predicting internet network distance with coordinates-based approaches. In *In INFOCOM*, pages 170–179, 2001.
[6] Jianping Pan, Y. Thomas Hou, and Bo Li. An overview of dns-based server selections in content distribution networks. *Computer Networks*, 43(6):695 – 711, 2003.
[7] Bernard Wong, Aleksandrs Slivkins, and Emin Gün Sirer. Meridian: a lightweight network location service without virtual coordinates. *SIGCOMM Comput. Commun. Rev.*, 35:85–96, August 2005.