CPSC 490 Senior Project: Anticipating Changes in the Data Plane

- Student: Charles Proctor
- Adviser: Professor Yang

Background

Typical software-defined networking (SDN) applications read from and write to the data plane with great frequency. Such applications often have complex and intertwined data-dependencies.

For example, suppose we want to compute the shortest path between two hosts in a network. The application must

1. read link information from a data store,
2. compute the shortest path between the source and the destination,
3. write the resulting path for use on incoming packets.

The written path is only valid so as long as the input data remains unchanged. Unfortunately, typical computer networks are highly dynamic. Each time the data changes, the shortest path function needs to be rerun. Before FAST, detecting data changes, re-executing functions, and maintaining state was the job of the SDN programmer.

As an application designed and implemented by the Systems Networking Lab (lead by Professor Yang), FAST listens to changes in the data plane, re-executing functions when their input data changes. Specifically, the FAST interface exposes a function store and a data access API.

SDN programmers register functions with the function store, while each function uses the data access API to read / write data. Behind the scenes, the FAST runtime tracks and analyzes the data access patterns and the FAST scheduler organizes re-execution of the functions.

Large portions of FAST have already been developed. Over the past few weeks, I have assisted in the preparation for the OpenDaylight summit, where the basic functionality of FAST will be demonstrated and taught to an audience in Seattle, WA.

Motivation

As described above, whenever the data on which a function is dependent changes, the function needs to be re-executed. In many cases, this can be an expensive computation. In the shortest path example, whenever a link fails (or loses bandwidth), the shortest path needs to be re-computed.

Furthermore, we make the following observations:

- When routing packets, real-time CPU computation should be avoided whenever possible. Although flow rules and other network data can be safely distributed across a series of hosts, function execution must often occur by some “central controller” that has access to the most recent data and can coordinate execution across the entire system.
• In contrast, quickly-accessible memory is readily-available, allowing the system to store a finite number of pre-computed results. As mentioned before, this memory is often distributed across a series of hosts. For example, flow rules are written directly to the switches. Therefore, whenever possible, it is more efficient to lookup a pre-computed result, rather than performing the computation in real-time.

• CPU cores are often under-utilized. Since networks are designed to handle peak capacity, the system must be built with hardware to route requests at peak bandwidth. That being said, when the network is under-utilized, the CPU cycles are wasted.

Combining the under-utilization of CPU cores and availability of cheap memory with the desirability of quick real-time lookups, these observations motivate the design of a system to pre-compute results under hypothetical data changes. As is detailed in the following section, this is the system I propose to design and implement for my senior project.

Design

At first glance, this problem seems computationally intractable: Given continuous-valued data, there are infinite possible data changes, all of which would need to be computed. To address these concerns, the following approaches will be taken, each of which is detailed in a later section:

1. Prioritize important customers.
2. Analyze data variation.
3. Ignore insignificant fluctuations.

Prioritization of Important Customers

In this context, a “customer” can refer to several different ideas.

• First and foremost, each function can have a different level of prioritization. For example, a routing function should run with top priority, whereas a network analysis function should run with lower priority.

• On a different level, certain function inputs should be prioritized over others. For example, suppose the path from A to D is consistently transmitting 10Gbps, while the path from A to F is only transmitting 100Mbps. In this case, a path routing function should prioritize pre-computation for (A, D) over (A, F), since the results from (A, D) are used on 100x the number of packets.

Analyzing Data Variance

By analyzing past data variation, the system will predict probable changes. For example, if link A has never failed, we can assume within reasonable probabilistic bounds that it will continue to succeed for the immediate future. In contrast, if link B has failed every five minutes for the last two hours, the system will compute the value of the function instance with and without link B.
Ignoring Insignificant Fluctuations

When exact results are unnecessary, estimations can be made and insignificant fluctuations can be ignored. Again using the shortest path example, if link A can support 10 Gbps while link B can only support 100Mbps, the system should not re-compute the shortest path for +/- 10Mbps changes in the bandwidth of A. In this context, we use estimation to determine the relative importance of data changes.

Obviously, there are more advanced applications of estimation techniques. For example, linear regression could be used to predict the values of a function, supposing the function were linear. Since the applications of such models requires harsh assumptions about the nature of the underlying function, I likely will not pursue this course.

Deliverables

By the completion of the semester, I plan to extend the base implementation of FAST to pre-compute function return values. Specifically, I plan to explore (at least) the three techniques described above.

As I explore various techniques for pre-computation, I will analyze their performance across a variety of different simulated computer networks. By the end of the semester, I will produce a written report comparing the performance of the different methods.