Finding Fault Location: Combining network topology and end-to-end measurments to locate network problems?

Chris Kelly - chris.kelly@oit.gatech.edu
Research Network Operations Center
Georgia Tech Office of Information Technology

CS8803 - Discrete Algorithms Semester Project

December 12th 2006

Abstract

Typically, network monitoring is from a central point of view: problems only noticed by small sets of end users may never be noticed centrally. End-to-end measurments can address this problem, but in most implementations it is left to the network administrator to correlate alarms and locate the problem. This paper explains my implementation of a tool, FaultFinder to partially automate this process by combining network topology information with end-to-end performance data on a campus network to locate network paths or nodes that seem to be causing trouble.

Keywords: communication networks, network monitoring, network performance, binary tomography, network tomography

1 Overview

Typically, network monitoring is from a central point of view: problems only noticed by small sets of end users may never be noticed centrally. End-to-end measurments can address this problem, but in most implementations it is left to the network administrator to correlate alarms and locate the problem. This paper explains my implementation of a tool, FaultFinder to partially automate this process by combining network topology information with end-to-end performance data on a campus network to locate network paths or nodes that seem to be causing trouble.

2 Topology Information

Ideally, I would like to take advantage of the finest granularity of topology available: every single physical device on the path, but often this will not be available because of the access to the network that it requires. With this in consideration, FaultFinder was designed to be flexible and work with any type of topology data collected varying from a layer 2 traceroute, to a standard layer 3 traceroute, to higher level network diagrams. Currenly FaultFinder relies on manual entry of topology into a database but any other tool could be used to automate the insertion of topology information into the database because it is stored in an easy to comprehend format.

Topology information is stored in the database in two tables: a table of points and of links. The point table indicates if a node is an endpoint or not, and the link table lists the direct connections from a node to another node. I also generate a table of path components which lists any two nodes and all of the links that must be traversed to get from one to another. The path component table is used to facilitate quicker calculations of path overlaps and must be recomputed each time the topology changes.

The topology is accessed in three ways:

- Loading into a data structure by iterating through all of the nodes and edges which lets FaultFinder use existing code to perform graph calculations and render images
- A database query enables FaultFinder to see which path components are in the paths between pairs of nodes. (select from components where either start or end = one of the endpoints, group by linkid, count, where count = # of pairs)
- A database query enables FaultFinder to see which pairs of endpoints have a path that run through a given segment. (select from component where linkid = the link id)

3 Performance Information

FaultFinder is usable with any set of end to end measurements for a topology. Selected examples include latency data (collected with smokeping) and throughput data (collected with iperf). This information is stored in a database table with a timestamp, the source and destination nodes, and the performance data. (An existing system, CPR (http://www.rnoc.gatech.edu/cpr/), collects and stores this data.) The initial implementation of FaultFinder assumed a binary

performance value of down or up, but the current implementation can deal with variation from a threshold as long as the threshold can be quantized before using the tool.

4 Building the Performance Model

The initial implementation of FaultFinder used a rating system for each node and link that compared the number of failed tests through that particular node or link with the worst case number of failed tests on any node or link in the entire topology. This was used because it did not require a failure threshold to be given to the tool, and the hope was that as bigger problems were fixed, FaultFinder would compensate and start showing smaller and smaller problems. This seemed to make sense but after some experimentation, I found this method to mask all problems when failure rate was consistent across nodes and links experiencing problems. It also would mask nodes and links experiencing a moderate failure rate when one node or link had a very large or complete failure rate. It's important to show all failures but with an emphasis on larger problems, so this wasn't acceptable.

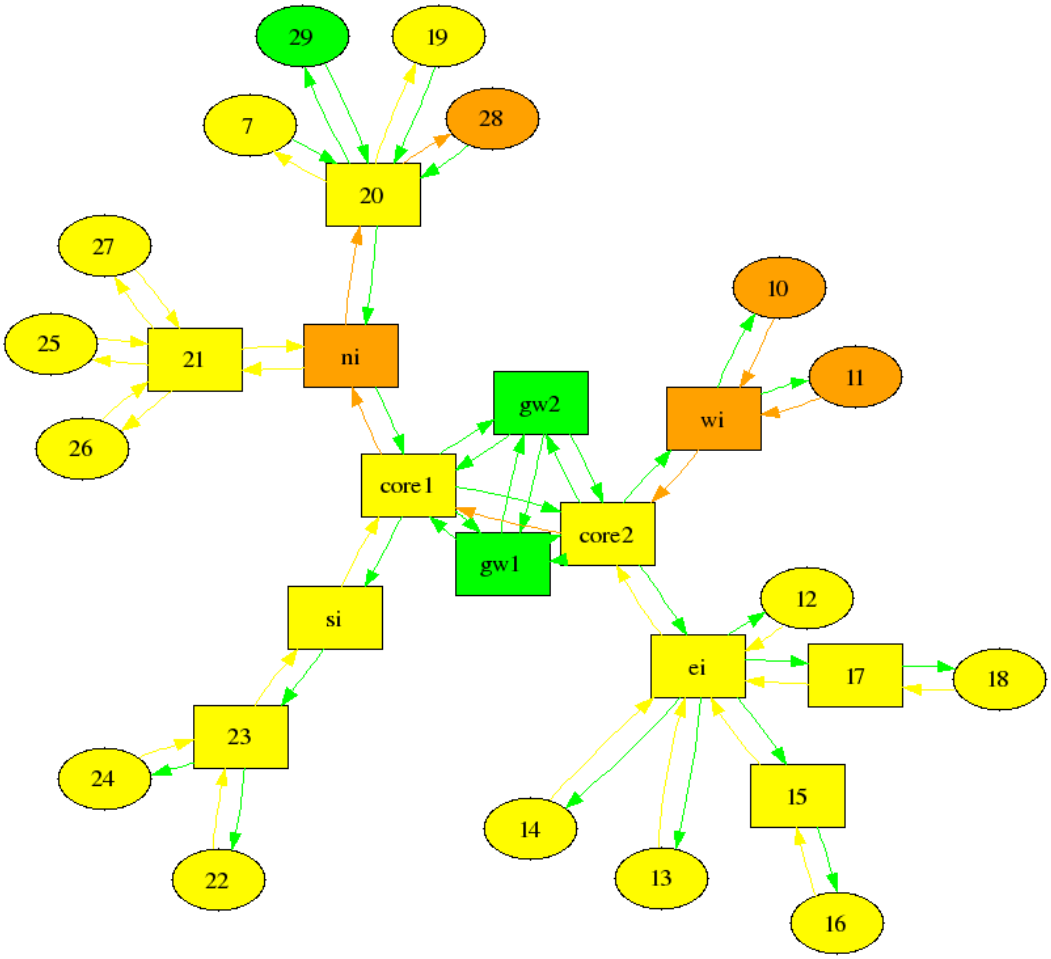
The current implementation of FaultFinder instead uses a percent failure rate per node and link. At the beginning of execution, the user provides a threshold for what should be considered a failure. This threshold is compared to the percent failure test data in the database, and a percent failure rate is calculated by dividing the number of tests above the threshold by the total number of tests through that node or link. This gives a good indication of links that are performing poorly and allows operators to tune the program to a higher threshold as problems are resolved.

Currently under development, the borders around nodes and the width of links will be scaled on a logarithmic scale by the number of tests running through the node or link. This will allow a quick glance at the output of FaultFinder to see which important links are experiencing problems.

5 Examples

5.1 Proof of concept on binary data on a fictional topology

A fictional topology was created with fictional performance data. In this example, 25 of the 256 performance tests fail and the rest succeed. The following graph was produced by the initial implementation of FaultFinder indicating the poorly performing network links:



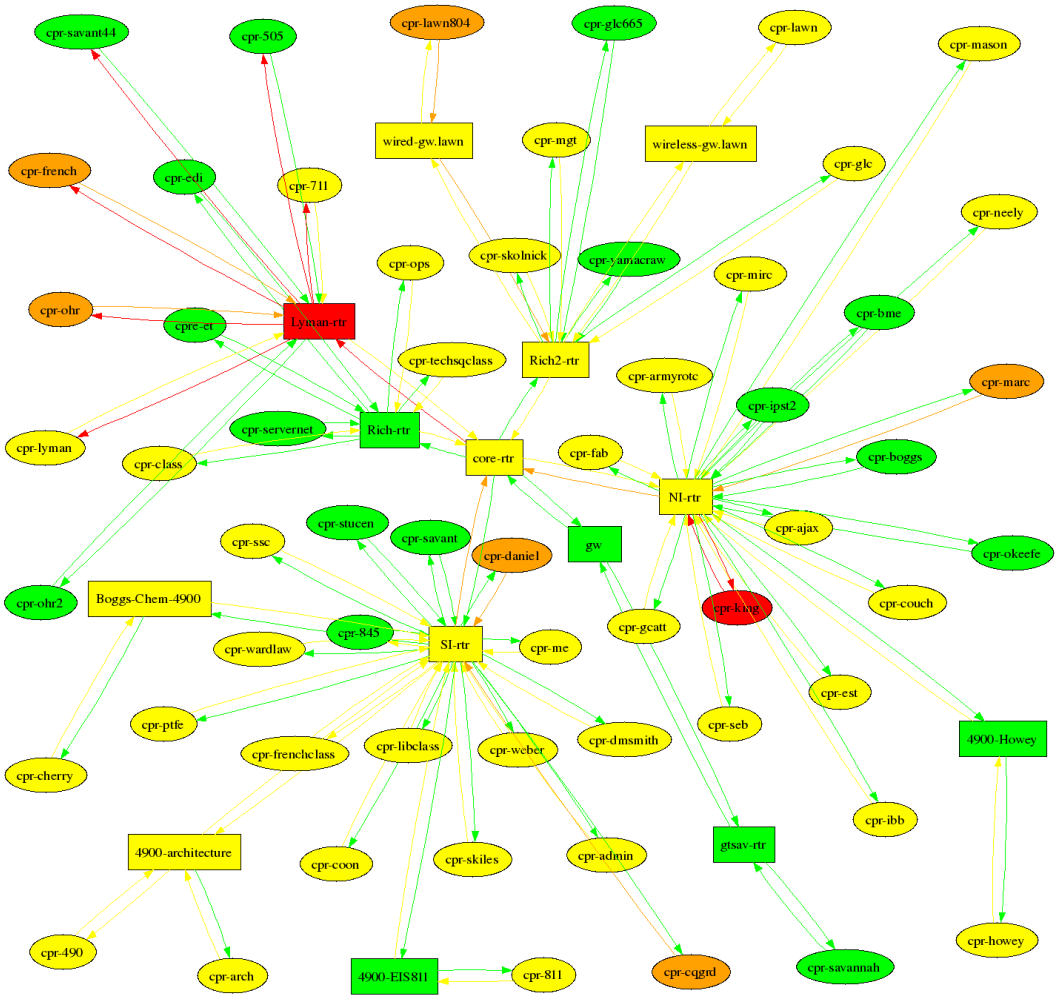
The worst performing path components are shown by this list of ratings:

- 11 to 9 is 0.0606909430438842
- 10 to 9 is 0.0606909430438842
- 4 to 3 is 0.120448179271709
- 9 to 4 is 0.1531279178338
- 6 to 20 is 0.209150326797386
- 3 to 6 is 0.26984126984127
- 20 to 28 is 0.583566760037348

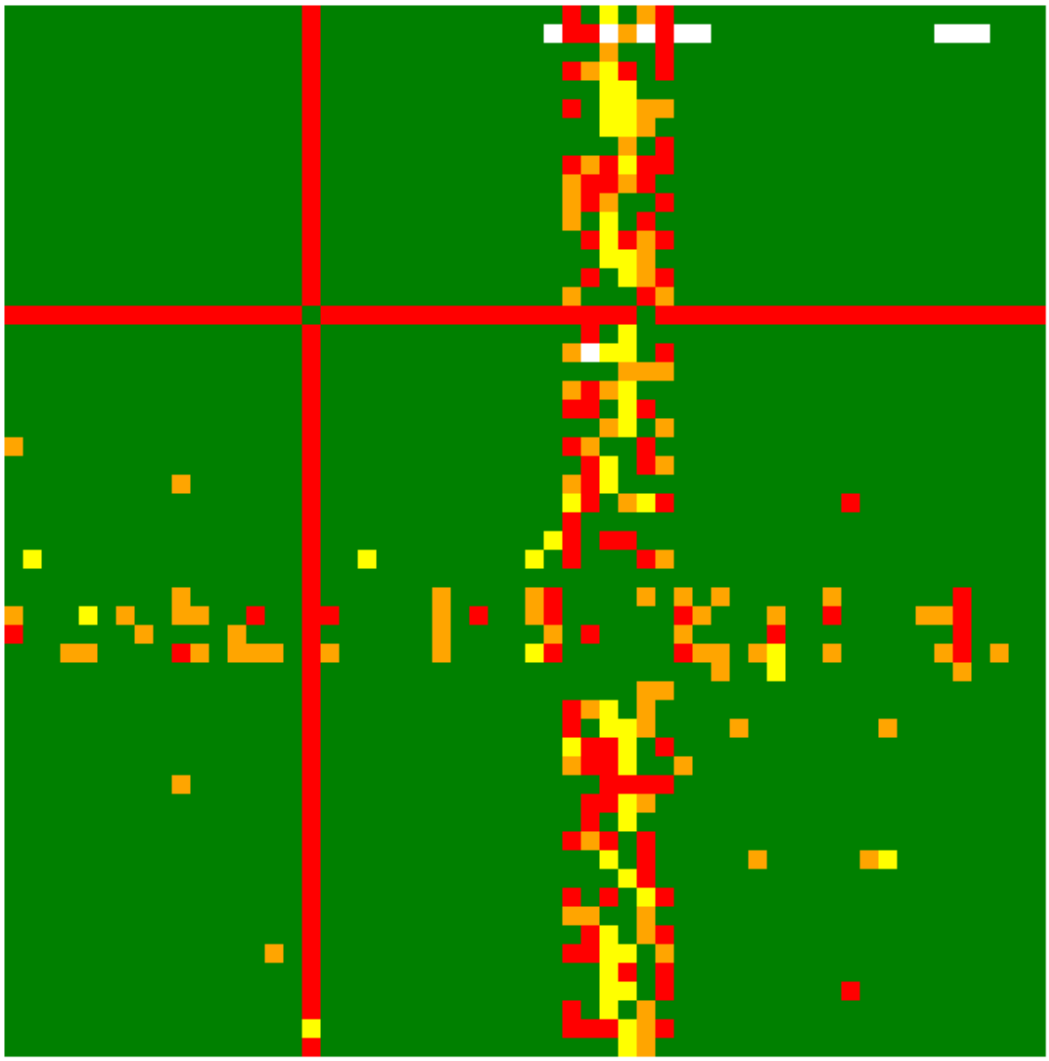
5.2 TCP throughput on the Georgia Tech campus network

This grah represents TCP throughput performance measurements (collected with iperf) from the CPR monitoring mesh of 80+ machines. The current implementation of FaultFinder was used with a threshold of 80Mbps was used to indicate success. The graph points out that there is a major performance problem with the Lyman router and the CPR monitoring node

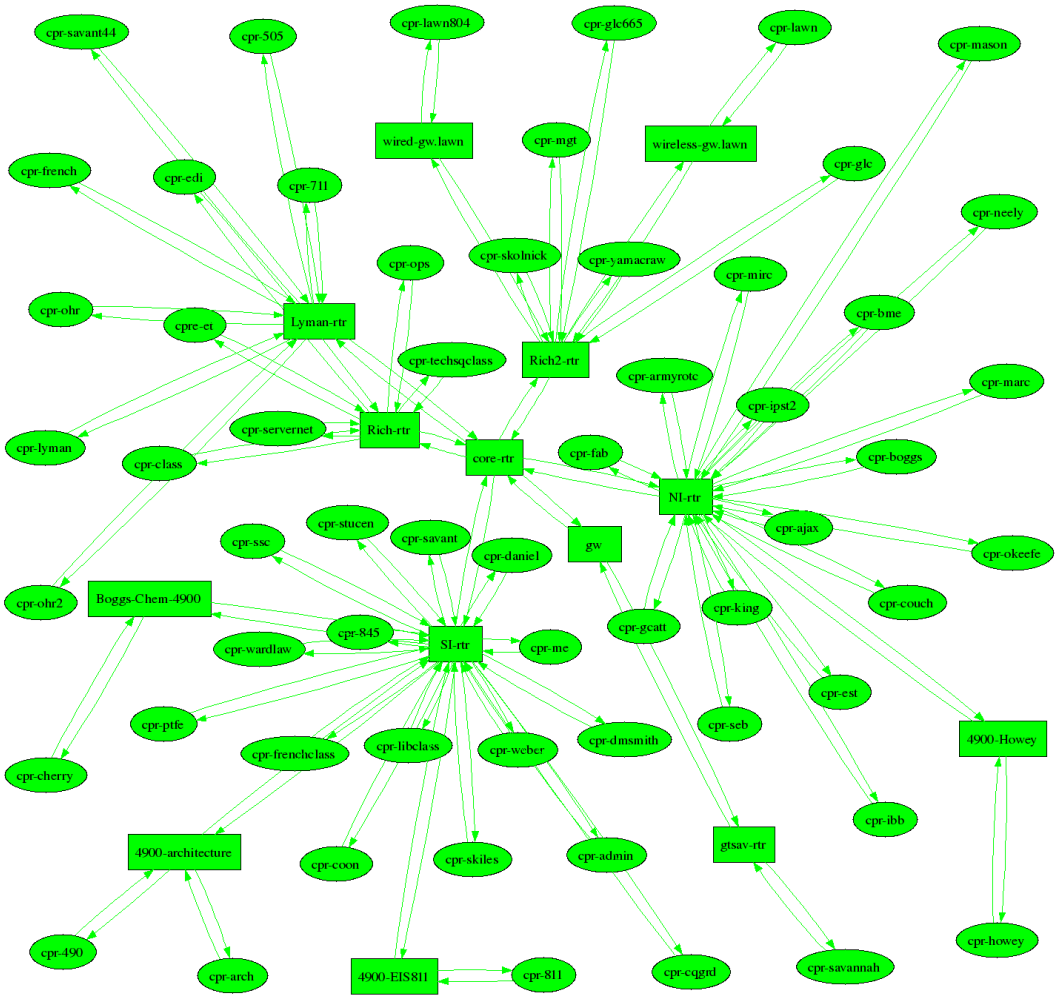
in the King building. The King building could be a machine or link failure, it's tough to say (It ended up being a 10mbps switch in that building!). As for the Lyman router, severe performance problems are indicated for all of the links in the direction from campus to the monitoring nodes attached to it. This indicates that something is probably going wrong with the router. After using this tool to find this problem, I reported it and a software update to the Lyman router resolved this problem as indicated in the third graph below.



Before the creation of this tool, I manually ordered rows and columns in a table based on default gateway for machines and generated the following graph, but as you can see it is a bit hard to infer some of the details. (The original had from/to/throughput information with a mouse hover. It was good for a general overview of the network, but hard to narrow down problems)

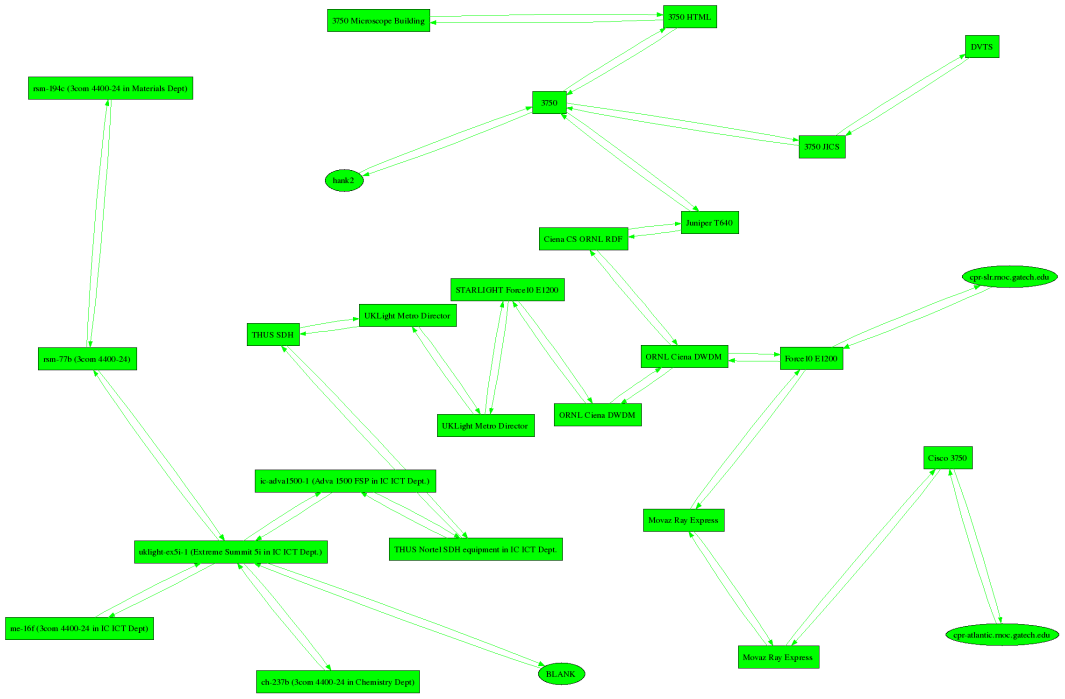


After resolving the issue with the Lyman router, a rerun of iperf on the CPR monitoring mesh indicated that something wrong with a pair of hosts and their throughput to all other hosts. These two machines were virtual machines on the same piece of physical hardware which apparently couldn't handle the IO throuput. I placed each of these machines on their own hardware and after another run of iperf, the network was finally all green. Increasing the performance threshold to 90Mbps also indicated no failures, and given the nature of the tests, any value over 90Mbps on a campus network is considered acceptable throughput.



5.3 Reachability on the ATLANTIC network

The ATLANTIC network is a fiber network that, among other things, connects Imperial College in London to Oak Ridge National Labs through Georgia Tech so that scientists at Imperial College can use a microscope at ORNL. This graph represents reachability data collected with ping on this network set on the Layer 2 topology. Given the small number of endpoints (only 4!) and the great overlap between them, this tools is not currently as useful for this network, but as more data is collected on this network using different measurement techniques, FaultFinder may help to locate problems.



6 Related Works

There is not much other work that has been done in this specific area, but there are a few examples. The first of these is Duffield's IMC 2003 Paper: "Simple Network Tomography." In the paper, Duffield builds an algorithm that combines end to end measurements that either pass or fail with a network topology to produce sets of link failures that could explain the current failure conditions on the network. He shows that the smallest set of link failures is usualy the most accurate.

To contrast Duffield's approach, I am using a flexibile performance metric to indicate a failure (ex: less than 80Mbps throughput on a switched 100Mbps network) of a test. Currently this is chosen manually by the operator and is constant across all links, but in the future FaultFinder will support separate performance metrics for each pair of nodes running tests based on past test history that was considered successful. Our approach also doesn't attempt to narrow down results to the simplest explanation: I instead determine a percent failure rate for each link segment and indicate those with the highest percent failure rate as the best candidates for further investigation. These approaches are similar, but I feel that my method (and it's future directions) may be more useful for situations where performance problems are not as simple as link failures and are indicated by past performance instead of a strict criteria.

7 Future Directions

There are many future directions for this work on FaultFinder and I intend to continue development of this tool for the foreseeable future. Here is a partial list of ideas to be

further explored:

- Test FaultFinder on more types of data collected with the CPR monitoring mesh on the campus network to see if I can find any more subtle problems that may have otherwise gone unnoticed. (Smokeping, pathload, etc)
- Utilize a Layer 2 map of the campus topology to achieve a much higher granularity in the paths. Some components that appear to be different at Layer 3 are actually on the same hardware so though they may only seem to be partially failing independently, the combined failure rate may be substantially larger.
- Investigate using some of the graphing tools from CAIDA (http://www.caida.org/tools/) which may allow for a much more comprehensible graph as the topologies investigated grow more complex.
- Create a method to automatically scale the output graph based on failures. Endpoints with no failures between them would appear to be directly connected to eachother and as problems are detected, the graph would show more and more details up to the level where it could single out a problem. (Example: in the ATLANTIC topology above with the existing topology, a single failure between hank2 and cpr-atlantic would result in a graph showing a problem between hank2 and ORNL Ciena DWDM, partial problems from ORNL Ciena DWDM to cpr-atlantic, and success from BLANK and cpr-slr to ORNL Ciena DWDM.)

8 Conclusions

Combining topology data with performance data can provide good indications of likely points of failure in the network. A visulization of this information like the one generated by FaultFinder is useful in allowing likely points of failure to be located with a quick glance. This graph and the average failure rates for points in the topology enable operators to only need to run manual tests to confirm likely problems and not to do the initial location which should save large ammounts of time and shorten the time to resolution for performance problems.

Advantages aside, there are several limitations of this approach that all require further investigation.

- Current display of one type of test on a graph (iperf for example) indicated several problems but once fixed, everything became green and stayed that way. Multiple performance metrics need to be aggregated together so that one graph can provide a comprehensive indication of problems.
- It is currently difficult to tune FaultFinder to prevent false positives due to generalization of what indicates a failure of a test. Future work should prevent operators from having to manually scale thresholds to try and focus on problems.
- Successful usage of FaultFinder requires a knowledge of the topology and good coverage of monitoring nodes. There isn't really a way around this and thus this tool will not be very useful for those without either.

- Multiple paths possible for tests would make results a lot less accurate. Because a shortest path algorithm is used that considers all the links to have the same weight, paths chosen by FaultFinder would likely be somewhat inconsistent with the actual routes that the packets took. The ideal solution to this is to store routing information in the database as well and use it to generate the actual graph, but this substantially complicates the code and requires infrastructure to gather the routing information.