Progress Report
Week of July 20, 2020

Edris Qarghah

Enrico Fermi Institute
University of Chicago

Weekly Goals

- Learn about tomography strategies.
- Develop tomography of static network (e.g., with latency/packet loss).
- Ensure perfSONAR node distribution allows visibility into entire network.
- Generate, store and draw shortest paths between PS nodes.
- Stretch: Simulate network activity.
- Stretch: Generate time series data based on that activity.
- Stretch: Learn how this connects to anomaly detection.
- Stretch: Explore how the model can be applied to real network data.

Daily Log

Monday, July 20

- Network Analytics Discussion meeting.
  - Shawn suggested there may be a paper on network topology.
  - Shawn shared details about how the network is structured.
    - Points of Presence (PoP)
    - ESNet (originally for Department of Energy labs only, but became available to high energy physics labs associated with LHC1) connected to universities which are predominantly on Internet 2.
  - Shawn suggested that end nodes that have only one connection to the toy network should be PerfSonar nodes (otherwise, we would have no reason to traverse to them and wouldn't even know they were there.
  - Potential graph augmentation: Create two networks (ESNet and Internet2) and connect them.
- Read about Points of Presence, as that was a term I wasn't familiar with.
- Read about LaTeXmakefiles (and makefiles in general) and finally got a working one that will not require babysitting with every new weekly progress report.
- Worked on Shawn's PerfSonar suggestion.

Tuesday, July 21

- Spoke with Ilija about static network tomography. He suggested I:
  - Create a histogram showing the sum of latency across all hops between every pair of PerfSonar nodes.
  - Create a histogram showing the probability of a packet being lost between any two pairs of PerfSonar nodes (i.e., $1 - \Pi_n(1 - p_n)$ where p_n is the probability a packet is lost on the nth edge).
  - Remove a random edge and see the impact on the network.
    - Recalculate packet loss.
    - For any pair of PerfSonar nodes whose packet loss changed, plot new histograms.
    - Check the original shortest paths between all pairs that have changed to determine what edges they have in common.
    - Try to determine which edge was killed.
- Discussed future goals:
  - Add bandwidth (typically sampled every 5-6 hours).
  - Use bandwidth in conjunction with paths to calculate impact of broken edge.
- Create tomography .py and functions
- Sent in time for 7/7-7/22.

Wednesday, July 22

- Create time tracking spreadsheet.
- Learned about graphs:
  - Bridges are edges whose removal would increase the components in a graph.
  - Bridge-connected components are the components that would be created if bridges were removed.
  - Chain decomposition (didn't end up being relevant)
  - Edge boundaries are edges that connect a set of nodes with nodes not in that set.
  - Measures of centrality, most notably communicability betweenness centrality (credit to my wife for identifying this as a viable tool for ensuring PerfSonar dispersal).
- Developed method to disperse PerfSonar nodes:
  - Find all bridge-connected components.
  - If the number of components exceeds the PerfSonar count, create a new graph.
  - Calculate communicability betweenness centrality (cbc) for all nodes.
  - Sort the nodes in each component by cbc (lower betweenness means a node is on the periphery and is a better PerfSonar candidate).
  - Order the components from smallest to largest and the cbc of its first node (to address issues like when two components have size one but one lies between the other and the rest of the graph).
  - Make the first k nodes of each component into PerfSonar nodes.
    - k = 1 + round(desired perfsonar nodes - number of clusters) / total nodes)
    - Due to rounding, it is possible to not have assigned all perfsonar nodes by the end of this process, so we take the remaining ones from the next elements of the largest component.
  - Advantages of this approach:
    - All 1-connected nodes are perfsonar nodes (because they become separate components and have connectivity 0)
    - All components contain at least one perfsonar node (so there are no clusters in the network completely lacking one)
    - We guarantee the requested number of total nodes and perfsonar nodes without having to add additional edges.
    - The use of cbc insures that the node connected to the bridge in any cluster is never selected.
  - Outstanding issues:
    - If we have several an extended stretch of degree two nodes that end in a degree one node (i.e., they don't create a cycle), each one of those nodes will create a separate component. PerfSonar nodes cannot be adjacent, but that still means every other one of these nodes will be a PerfSonar node.

Thursday, July 23

- Read (Service Analysis and Network Diagnosis) SAND project proposal for a better understanding of how I factor into this objective: "Using machine-learning techniques, develop a performance anomaly detection service to issue alerts and alarms about detected end-to-end performance degradation incidents."
- Created and sent out When2Meet for SAND Working Session.
- Read more about measures of network centrality in the hopes of finding a faster alternative to betweenness.
- Factored PerfSonar node generation out of initial graph generation.
- Stored PerfSonar edgelist on Graph.
- Created function to load saved graphs (to save time, because graph generation was getting expensive)
- Found shortest paths between perfsonar nodes.
- For each edge, store the paths it participates in.

- Color-coded edges along paths between PerfSonar nodes.
  - Non-path edges are black and narrow.
  - If an edge is part of multiple paths, it has an edge coloring for each one.
  - The edge color and style cycle to allow differentiation of 40 different paths.
  - Path coloring on an edge after the edge is initially drawn is semi-transparent.
  - You can have variable width for edges or variable style (i.e., dotted) but not both.
  - Read a LOT of documentation about colormaps, style cycling, changes in how matplotlib handles colors, drawing edges in NetworkX, etc.
  - Determined latency and packetloss along routes and plotted them in histograms.

Friday, July 24

- Finished reports.
- Read about Biconnected components which were a consideration I did not take into account.
- Got access to the SAND Google group and read up on citable code.
- Used Google Scholar to search for literature on network topology.

Achievements

I learned about...

- Makefiles
- Points of Presence
- SAND Project (read proposal)
- Existing topology research
- Basic static network tomography
- Bridges, articulation points, edge boundaries, bridge and biconnected components
- Measures of centrality (i.e., Communicability Betweenness Centrality)
- How NetworkX and Matplotlib are conspiring against me (aka, edge drawing wonkiness and style cycling limitations)

I created...

- More robust networks:
  - Ensured degree 1 nodes were PerfSonar nodes or removed.
  - Rethought this and decided to use bridge-connected components instead.
    - Ditched networks that didn't have enough ps nodes for the components
    - Distributed nodes proportional to component size
  - Used Communication Betweenness Centrality to ensure edge boundaries are not selected to be PS nodes.
- Better visualizations of networks (differentiating edges is HARD):
  - Changing color, edge style and width (but not all three)
  - Drawing edges separately
- Basic tomography histograms

Roadblocks

Questions

- How do I find a packetloss rate balance that doesn't kill the network?
- How should I handle articulation points?
- How should I handle components that are really only a throughput to a PS node (ideally, these would be omitted from PS creation; there's a possible easy fix for degree 2, more difficult otherwise)?
- How should I handle drawing changes in paths?

Problems

- Connectivity is expensive to calculate and slows down graph generation dramatically.
- NetworkX has bizarre limitations on when/how it allows the styling of edges.
- Strings of degree 2 nodes that end in a degree 1 node are each considered separate components, which results in over assignment of PS nodes.

Challenges

- Making sure the entire network is traversable from PS nodes.
- Drawing paths on the network.

Plans for Next Week

- Further Static network tomography
  - Determining dead edges from impact on shortest path
  - Add bandwidth and measure bandwidth impact
- Draw ONE path at a time (see changes in a single route)
- Simulate network activity
- Measure activity from specific nodes/generate Time Series
- Develop tools for tomography and anomaly detection