K-Path Overlay Routing for Resilient Clouds

Bennett Ellis

Overview

- Interested in ways to create additional resiliency to network faults and partitions in overlay networks
- How?
- Multipath routing

Agenda

- K-Path Routing
- My Implementation
- Testing My Implementation
- Demo
- Practicality

K-Path Routing

- Looking for a set of k edge or vertex disjoint paths in a graph such that the sum of the lengths of each path is the minimum of the corresponding sums of all possible sets of k edge or vertex disjoint paths for that graph.
- Possible Algorithms: Suurballe's, Algorithms from *Survivable Networks* by Ramesh Bhandari, Min-cost/Max-flow

Suurballe's Algorithm

- Run Dijkstra's to get shortest path tree.
- Use cost of each edge in tree to modify weights of every edge in the graph to get a transformed graph
- Rerun Dijkstra's to get shortest path to destination.
- Repeat 2-3 as necessary (1 repetition per desired extra path)
- Remove overlapping edges in paths to destination from different iterations of Dijkstra's

Algorithms from *Survivable Networks*

- Run Dijkstra's to get shortest path
- Replace each edge in path with a negative edge in the opposite direction
- Rerun modified version of Dijkstra's
- Repeat steps 2-3 as many times needed (1 repetition per desired extra path)
- Remove overlapping edges from paths generated by each Dijkstra's.

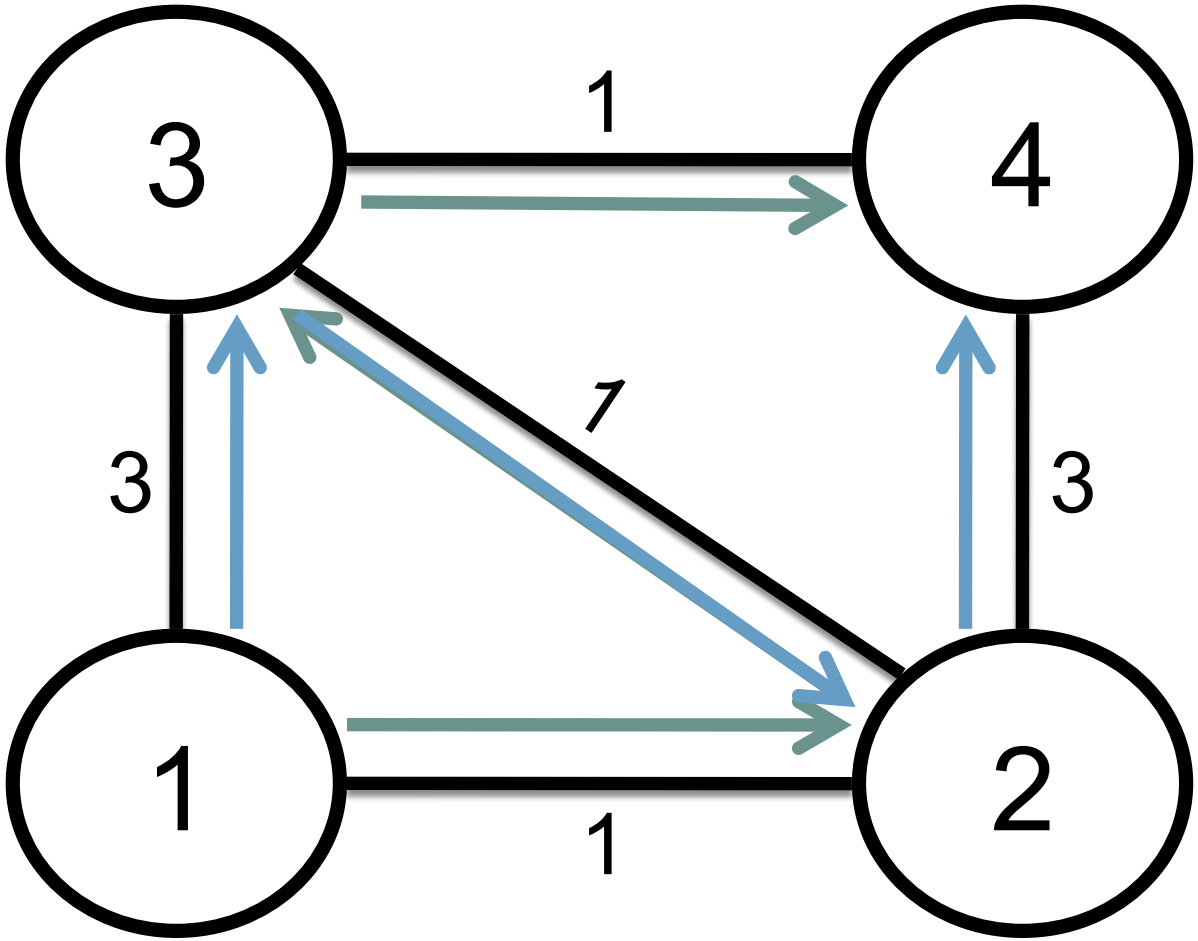
Min Cost/Max-Flow

- Recast problem as a flow network. Each edge has a capacity of 1 unit of flow and a cost has a cost per unit of flow for using that edge.
- Variation of Edmonds and Karp
  - Only increment flow by 1 in each iteration
  - Use Dijkstra's to find shortest augmenting path based on cost rather than finding shortest augmenting path based on number of hops
  - At the end: look at edges with positive flow
- K iterations will find a minimum cost path for sending K units of flow, which is equivalent.

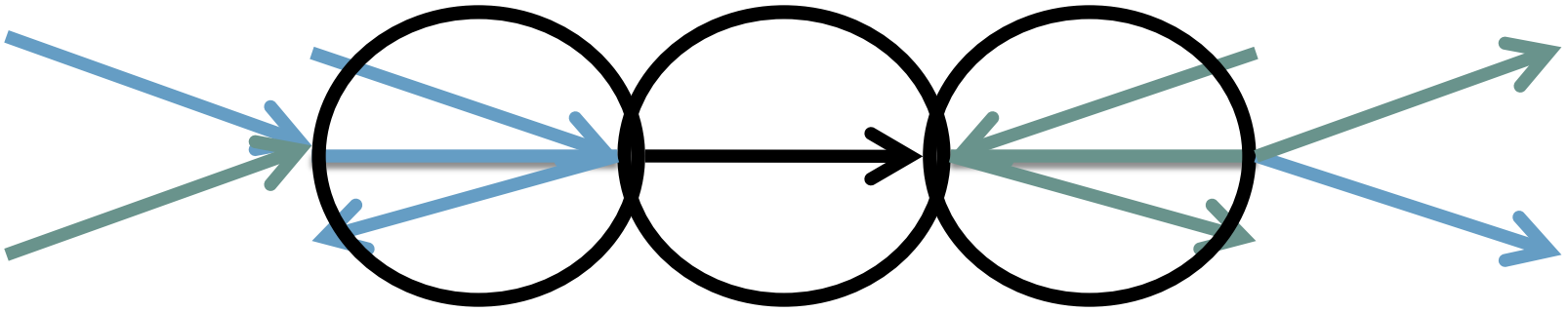
Algorithm Comparison

- Each runs K iterations of Dijkstra's
- Surballe's requires a modification of each edge in the graph after every iteration
- *Survivable Networks* requires the reversal and negation of edges of each iteration's path
- Min-Cost/Max-Flow requires modification of flow of the path found in every iteration
- Decided to go with Min-Cost/Max-Flow

Min-Cost/Max Flow Illustration



Edge to Vertex Disjoint



My Implementation

- Decided to do it as a model of an overlay network written in C.
- Entire network known to each server in model
  - N servers, each numbered 1-N
  - Bidirectional links (max one per pair of servers)
- UDP Communication
- Clients on same machines as Servers
- Source Based Routing
- Small buffer at exit node of model

Source Based Routing

- Routing info stamped into packet header for packets between servers
- 1 bit per link in the network
- Each server calculates same global ordering of each edge => same unique 1 bit mask calculated for each edge

Small Buffer

- Potentially receiving multiple copies of each message, out of order.
- Add sequence number to each packet based on source server. End server buffers packets based on source server.
  - Upon receiving each message
    If in order: deliver and then deliver contiguous buffered messages
  - If out of order: buffer
- Flush buffer every 10 ms (or if it reaches max # of packets it will buffer from one source)

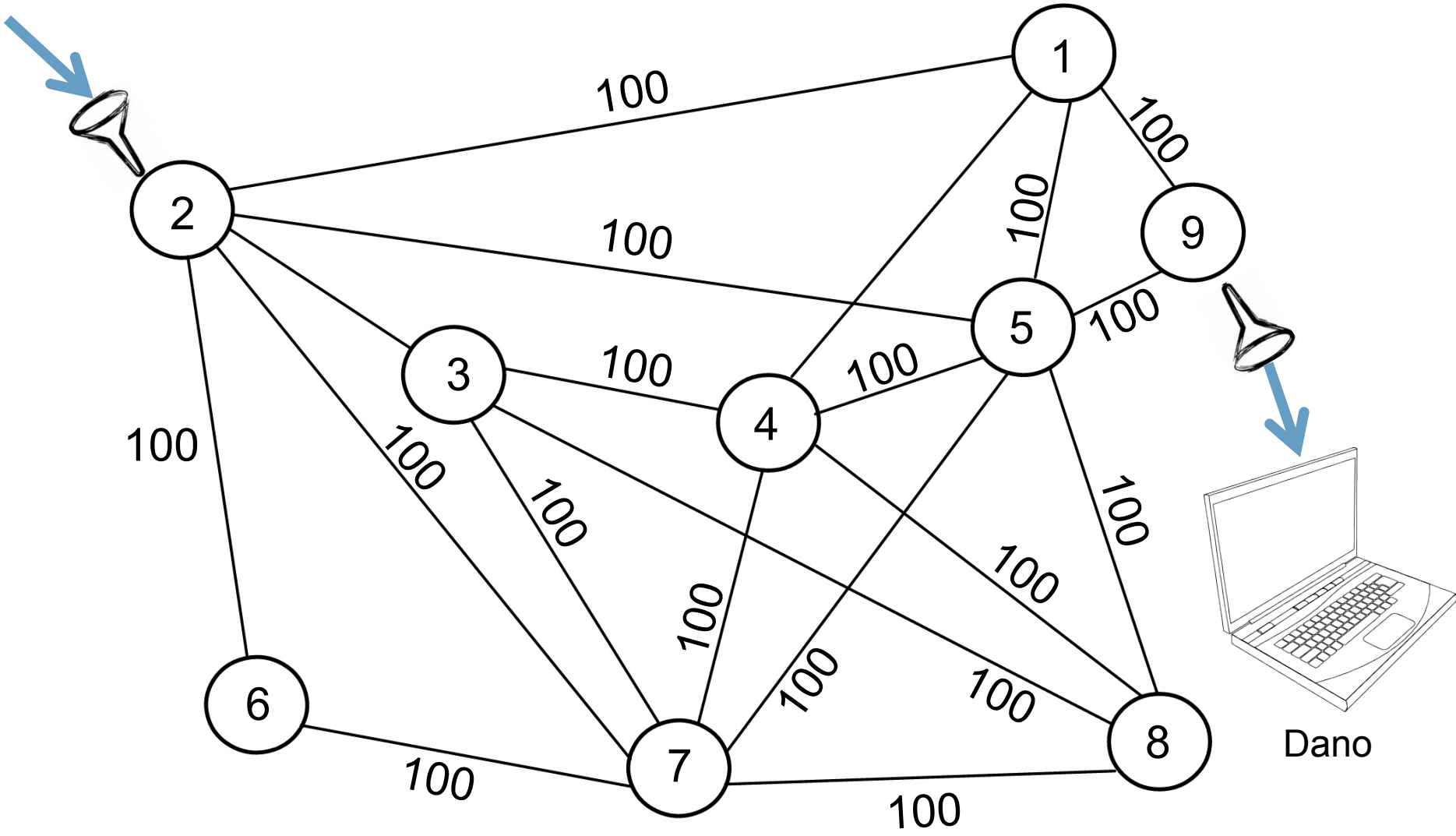
Testing

- Implemented naiive routing algorithm as well as a chat client. Used Simple network
- Implemented Min-Cost/Max-Flow algorithm and tested chat client
- Implemented funnels to stream larger quantities of data over the model

Demo

- Wrote a funnel client
  - On entry node in overlay: listens for outside packets on a port, encapsulates it in in appropriate header, passes to server along with request to deliver to a particular exit node using a particular number of paths
  - On exit node: listens for output from exit node server, strips headers, sends packets to an IP address set up in advance.
- Added Statistics
  - Entry node: picks out paths chosen every 1000 packets
  - Destination node: prints path packet actually took every 100 packets and time spent in overlay in microseconds
- Dr. Amir kindly let me test this on his cloud with a video stream.

Demo Graph



Practicality: Positives and Negatives

Negative	Positive
Pay K times per packet	Occasionally willing to do so for increased reliability
Requires knowledge of entire network	Common situation in Overlays
Model only recognizes clients on the same machine as servers	Only because model doesn't have info to find appropriate exit node based on outside destination IP. Fix by porting to a real overlay.
Model doesn't do any packet replaying between servers.	Again, only a limitation of the model. Port to a real overlay.

Questions?