EGOIST in the Wild

November 12, 2007

The purpose of this project is to develop a fully distributed system that supports the basic functionality of EGOIST, i.e. best response wiring, under churn, without the assistance of a centralized bootstrap server. A peer v_i connects (with directed links) to k_i peers in the overlay network, and is responsible for knowing the list of peers that participate in the overlay network, and for reporting any failures or departures of peers that it detects. You will implement an agent that runs in each peer and contains two basic components:

- **Link State Protocol:**

  1. Each peer runs a link state protocol, and broadcasts the list of the peers that it is connected to, and the pairwise delays (in msecs), *e.g.*, if peer 1.2.3.1 connects to peers 1.2.3.3 and 1.2.3.8, it broadcasts:
     1.2.3.1 1.2.3.3 13
     1.2.3.1 1.2.3.8 19
     Each peer listens to the Link State messages so it can construct the list of all the peers in the overlay, and an edge representation (annotated directed links) of the overlay graph E.
  2. Each peer `ping`s all the other peers every T_1 seconds. If a peer is not responding, it broadcasts a message, to inform the other peers that this peer is down, *e.g.*, if peer 1.2.3.1 `ping`s, and finds that peer 1.2.3.7 is down, peer 1.2.3.1 broadcasts:
     1.2.3.7 1.2.3.7 0
     Peer 1.2.3.1 and every peer that listens to this message must delete peer 1.2.3.7 from the peer list and all the links to and from peer 1.2.3.7 in the edge representation E. If peer 1.2.3.7 becomes alive again or later re-joins the networks, it will broadcast its peers and the delays to these peers and thus will begin to participate in the system again.
  3. A newcomer peer in the network must connect to an active peer, download the list of peers and the edge representation E. It connects to k_i peers using Local Search (see below), and broadcasts the list of the peers it is connected to, along with the pairwise delays. From hereonin, it also participates in the Link State Protocol.

- **Local Search:** Every T_2 seconds, each peer runs the Dijkstra algorithm on the edge representation E, and calculates its cost to reach all other peers. It marks, uniformly at random, one of the k_i peers that it is connected to as a candidate for swapping, and checks if by connecting to any of the remaining $n - k_i - 1$ peers (n is the total number of peers that participate in the overlay) that participate in the overlay, it will decrease the cost to reach all the peers in the overlay. To do this, the shortest path cost to each peer has to be estimated. The peer connects to the peer that minimizes the aforementioned cost.

Note that there is no need for synchronization as each peer will operate independently of the others. In order to assist routing, you have to save the IPs of the immediate neighbors in `neighbors.dat`, the edge representation in `Graph.dat`, and the list of the participating nodes in the overlay in `nodes.dat`

Requirements: UNIX and socket programming in C and/or C++. You might find it useful to develop parts of the code in python or perl.

A Basic (non-distributed) implementation of the client in C can be provided to you to get you started.

For additional questions and clarifications, do not hesitate to contact
George Smaragdakis (gsmaragd@cs.bu.edu) and Mema Roussopoulou (mema@eecs.harvard.edu)