Advanced Topics in Computer Systems, CS262B
Prof. Eric A. Brewer

Making Gnutella Scale

March 30, 2004

I. Background

Huge effort to build structured P2P networks, in part to fix the poorly viewed file-sharing solutions...

- Structured => control over degree and hop count
- Maintenance required, especially under churn
  - often O(lg n) operations to repair after a node leaves
  - must rereplicate data and pointers to maintain invariants...
- Can do exact-match searches well, but general searches only poorly
  - file searches are just keyword matches, but these are a terrible fit for DHTs
  - especially awkward to map keyword searches in the presence of churn, since you must update the search structures (e.g. inverted index)
  - most searches are for “hay” -- i.e. popular content
- Very symmetric in general, doesn’t really exploit heterogeneity in nodes or queries

Unstructured solution has minimal maintenance, but uses floods for search and still may not find things...

What if you just ignored DHTs and fixed Gnutella directly?

When do you need structure?

II. Gia: Basic Approach

Things you need to keep:

- self-scaling: adding more nodes increases capacity
- make use of supernode-like heterogeneity

New things:

- random walks rather than floods for searching
- flow control on queries
- improving topology over time
- one-hop replication: meta-data about your immediate neighbors (but not the data itself)

III. Topology

Track of set of nodes that you know about (> set of neighbors)

Keep 3-128 neighbors, less if you have low capacity (minimum allocation per neighbor)

Pick from this set periodically to find a new neighbor

- pick a random subset, pick the highest capacity node out of that subset
- ask that node, Y, “won’t you be my neighbor?”
    - if Y has room, then accept
    - if not, accept only if there is a current neighbor with less capacity, pick the one with highest degree, Z, and drop it (since it will be hurt the least), and only if degree is higher than that of Y
    - need some hysteresis -- ensure that there is some delta between Y and Z before switching
- satisfaction level to figure out how often to explore topology changes
- seems find a set of neighbors relatively quickly (minutes or less)
- claim: high capacity nodes have more neighbors (but only one hop out?)

IV. Flow Control

Gnutella drops packets under overload

Better would be to prevent overload proactively using flow control

Basic idea: credit-based flow control

- one token for each simultaneous query you can handle (= capacity)
- allocate based on relative capacities of the neighbors
    - the higher capacity you advertise (to accept), the higher tokens you receive for your own outgoing queries
    - allocation analogous to lottery scheduling
- must have token to do a query

V. Search Protocol

Biased random walk

Choose the neighbor with highest capacity for which you have tokens

TTLs to bound length of walk. Also stop when you have enough responses

Don’t reuse edges (but you can reuse nodes) during a walk

Route answer back to issuing node (why?)

Key point: for popular items, a random walk is sufficient and cheaper!

VI. Performance

2-3 orders of magnitude better

All four changes contribute, but the biggest winner is one-hop replication (given random walk!)