Develop a Fast Approximate Shortest-Path Algorithm for Large-Scale Network

Research project

Author: Mikhail Anoprenko, group 185
Supervisor: Oleg Sukhoroslov, Docent, HSE
Curator: Maxim Prikhodko, Lead Key Projects Engineer, Huawei

Problem statement

- Routing problem in computer networks
- Network is a directed weighted graph G with n vertices and m edges
- Want to compute forwarding table T_u for each node u, $T_u(t)$ is the next hop node for u towards destination t
- Each node knows the whole topology of the network and computes its forwarding table independently

Requirements

- Solution must be loop-free in distributed model
  - For any source and destination a packet must be eventually delivered
- Solution should produce short paths
  - Minimize average relative stretch of paths between all pairs of nodes comparing to the shortest possible paths
- Solution should be fast
  - Minimize average run time of the algorithm in each node comparing to the basic solution

Basic solution

- Each node runs Dijkstra's algorithm in order to build shortest path tree
- Loop-freedom and zero paths stretches are achieved simultaneously
- Needs a heavy data structure and scales poorly on graphs with tens of thousands of nodes
- Widely used in OSPF and IS-IS protocols

Known approaches

- Optimizations for social or geographical graphs
- Optimizations for paths search with single source and single destination
- Incremental SPF

Evaluation and testing

- Testing framework supporting:
    - Graph generation
    - Algorithm evaluation
    - Validation of correctness
    - Measurement of path stretches and run time
    - Basic solution using std::priority_queue
- Used graph topologies:
    - Square grid
    - Multi-homed network
    - Fat-tree network
- Used normal and discrete distributions of edge weights

Algorithm: BFS-DAG

- In node u run breadth-first search, compute $d_1(u, v)$ for each v
- Consider only edges $x \mapsto y$ where $d_1(u, x) < d_1(u, y)$, the form a DAG
- Find the shortest path tree rooted at u in this DAG in linear time using dynamic programming
- Decrease of d_1 along the forwarding path implies correctness
- Complexity: $O(n + m)$, heavily outperforms Dijkstra

Algorithm: Clustered Dijkstra

- Divide nodes into clusters. Split problem into intra-cluster routing and inter-cluster routing
- Clustering must be consistent among all nodes
- Each node should only build forwarding table for its cluster and the global inter-cluster forwarding table
- These tables are combined in order to obtain a forwarding table for the whole graph
- Any algorithms for clustering and routing may be combined

Clustered Dijkstra: implementation

- Greedy clustering: consider edges by weight ascendance, contract the edge and unite two clusters, limit cluster size by a constant
- Use the basic solution for both inter-cluster and intra-cluster routing

Evaluation results

- BFS-DAG is very efficient in practice
  - Run time is less than Dijkstra's by at least 5 times in almost all cases
  - Average relative path stretch never exceeds 4%. Almost zero for normal weights distribution
- Clustered Dijkstra may be efficient in theory
  - May have less time complexity in case of efficient clustering
  - Appeared to be ~~a complete rubbish~~ inefficient in practice

The End

Questions?