Efficient Precomputation of Quality-of-Service Routes

Anees Shaikh   Jennifer Rexford   Kang G. Shin

University of Michigan   AT&T Labs – Research

July 8, 1998

Routing in Integrated Services Networks

- Support the migration of voice, data, and multimedia services to an integrated backbone network
- QoS routing considers network load and flow traffic spec
  - Optimize usage of network resources (balance load)
  - Satisfy the QoS requirements of each flow
- QoS routing consumes bandwidth, CPU, and memory resources
  - Exchange recent information about link load
  - Compute suitable route for each flow

How can we control *overhead* in large networks without degrading *performance*?

Outline

- Model of source-directed link-state QoS routing
- Precomputation of QoS routes
- Performance evaluation results
  - tradeoff with on-demand routing
  - link-state update policy
  - effects of network topology
- Conclusions and ongoing work

Source-Directed Link-State Routing

Link-state QoS routing (e.g., QOSPF, PNNI)

- Source router computes/selects path for each flow
- Route based on link-state database and flow requirements
- Link state represents available resources
- Link state updated periodically or on large change

Parameters affecting network overheads

- Route computation frequency (CPU)
- Link-state update frequency (bandwidth, CPU)
- Complexity of route selection algorithm (CPU)
- Network size and connectivity (bandwidth, CPU)

high accuracy $\longleftrightarrow$ low complexity

Routing and Resource Reservation Model

Compute a route to destination with bandwidth b

- (Optionally) prune infeasible links ($util(l) + b > 1$)
- Compute minimum-cost path(s) to destination ($\sum_l c_l$)
- Path costs based on load and/or hopcount

$\Rightarrow$ cheapest, shortest (feasible) path

Hop-by-hop signalling

1. Test admissibility ($util(l) + b \leq 1$)
2. Reserve bandwidth ($util(l) = util(l) + b$)

Routing failure: source cannot find a feasible path
Set-up failure: resource reservation fails during signalling

Route Precomputation

Benefits of precomputed routes:

- Amortize computation overhead over multiple flows
- Simple CPU provisioning with periodic recomputation
- Reduced set-up delay by avoiding on-demand computation

$\Rightarrow$ trade-off between processing overheads and quality of routes

Practical considerations:

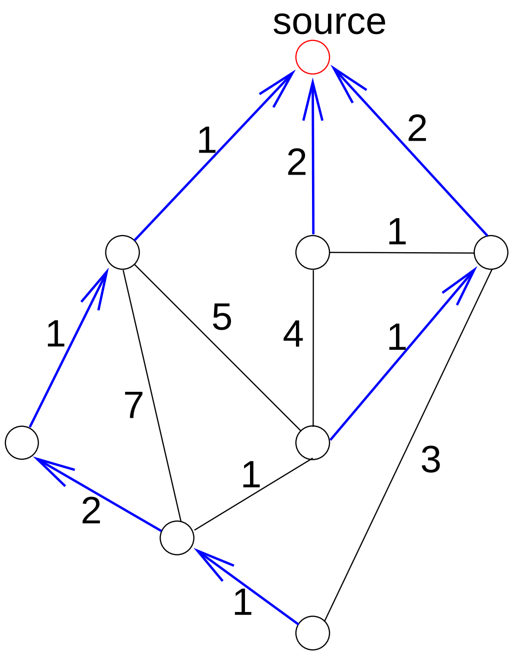
- How frequently should routes be computed?
- How should precomputed routes be stored?
- How much work to perform at flow arrival?
- How should multiple routes be computed?

Precomputation and Delayed Extraction

Key mechanisms:

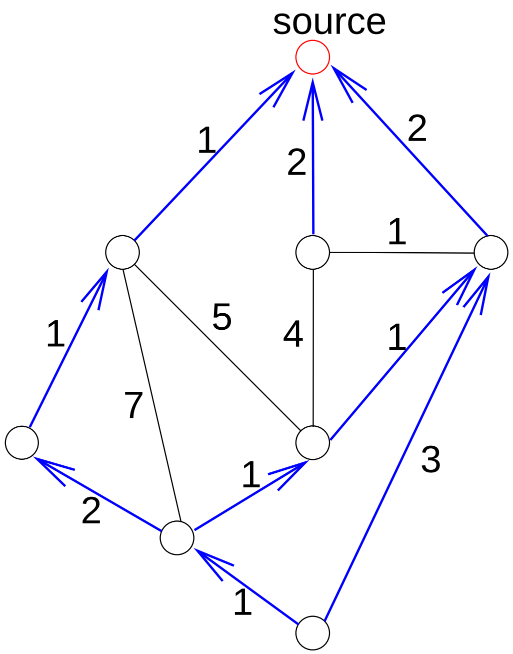
- Precomputation of minimum-cost graph
  - modified Dijkstra algorithm gives all minimum-cost routes
- On-demand route extraction with feasibility check
  - depth-first extraction using latest link-state information
- Coarse-grain link costs
  - reduces computational complexity of routing algorithm
  - increases likelihood of multiple minimum-cost routes
- Reranking of multiple routes during extraction
  - rerank links to implement alternate routing

Compact Storage of Precomputed Routes



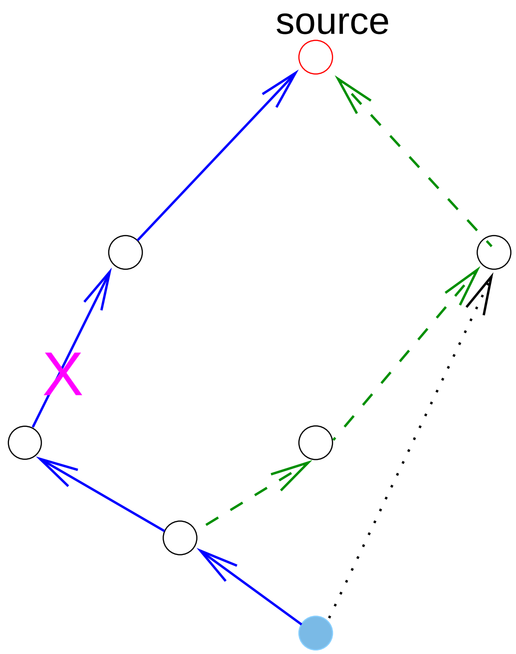
- Generate shortest-path graph
- Amortize calculation over all destinations
- Leave routes in shortest-path graph
- Avoid variable-length path cache

Generating Multiple Minimum-Cost Routes



- Alternate routes should have similar cost
- Near-shortest path algorithms too expensive
- Focus on simpler case of multiple *equal*-cost routes
- Increase "ties" by discretizing and mapping link costs

Delayed Extraction of Precomputed Routes



- Extract routes when flow arrives
- Traverse graph depth-first to find feasible route
- Check feasibility with latest link state
- Rerank parent pointers for alternate routing

Simulation Model

Event-driven connection-level simulation of link-state routing

- Routing: Minimum-hop least-cost route
  - link cost exponential in utilization
  - coarse-grain link costs ($C = 5$)
- Varied network topologies
  - sparse 19-node MCI backbone (avg. degree 3.4)
  - 125-node uniformly-connected network (degree 6)
- Uniform destination selection with Poisson arrivals
- Heavy-tailed flow durations
- Flows request bandwidth $2 - 3\%$ of link capacity

Metrics

- Blocking probability (overall and due to set-up failure)
- Per-node route computation rate

Configuration Parameters

- Traffic and topology
- Link-state updates
  - purely periodic
  - triggered on change in available bandwidth
  - with and without hold-down timer
- Background recomputation period
- Max number of signalling attempts (1)
- Enable triggered recomputation
  - trigger on routing or set-up failure
  - reextract on set-up failure
  - do not prune offending link on recomputation

Results and Observations

- Feasibility checking improves performance over path caching
  - significantly reduces network overhead by limiting set-up failures
  - benefits from having multiple routes
- Precomputation with triggered recomputation reduces overhead
  - computation frequency 10 times lower than on-demand
  - triggered recomputations dominate at high computation periods
- Feasibility checking remains useful with stale information
  - feasibility-check is competitive with on-demand when using triggered updates
  - fine-grain costs do not help much as staleness increases
  - triggered updates with hold-down timer maintains relative gains

Summary

Goals and principles

- Reduce route computation overhead
- Compute multiple minimum-cost paths
- Represent multiple routes in a compact form
- Apply most recently available link-state information
- Dynamically re-order precomputed paths

Enhancements and Ongoing Work

- Generate *near*-minimum-cost routes
- Consider *path cost* rather than link feasibility
- Evaluate bandwidth/traffic class-based precomputation
- Further exploration of link-level alternation
- Extend evaluation to wider range of topologies and workloads