# Configuration of Reserved Delivery Subnetworks

Ruibiao Qiu Jonathan S. Turner

Applied Research Laboratory
Department of Computer Science
Washington University
Saint Louis, MO 63130, USA

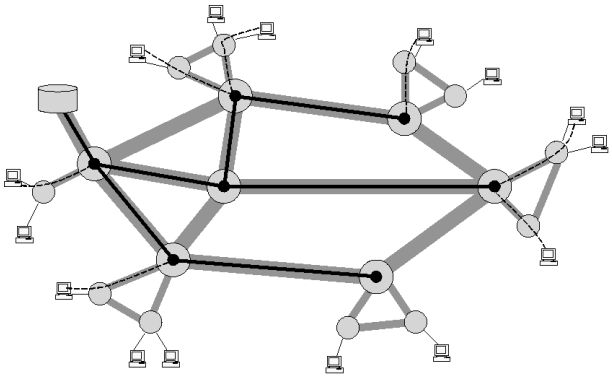
*Abstract*—**We introduce the concept of a *Reserved Delivery Subnetwork* (RDS) to allow an information service provider to deliver a higher quality of service to its customers. A network provider implements an RDS by provisioning reserved bandwidth on paths from a central site to distributed locations, where customers of the information service are located. The amount of bandwidth reserved is a function of the mean and variance of the traffic expected at the various locations. To configure an RDS, the network provider must select the links to be included and must dimension the reservations on those links appropriately. Network resource usage can often be reduced by routing flows destined for nearby cities along common paths. We show that the problem can be formulated as a minimum cost network flow problem with a concave cost function (one where the cost per unit flow decreases as the flow increases), which is a well-known NP-hard optimization problem. We introduce an approximate solution method and evaluate it experimentally. Our results are typically within a small factor of an easily computed lower bound.**

*Keywords*—**Reserved delivery services, minimum concave cost network flows, approximation algorithms.**

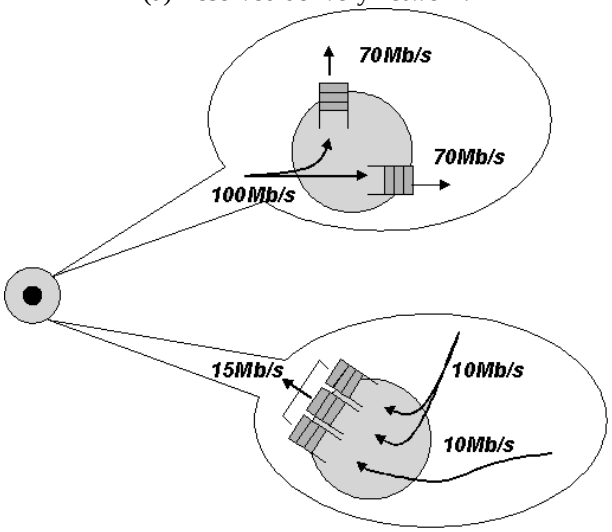
## I. Introduction

A *Reserved Delivery Subnetwork* (RDS) is a semi-private network infrastructure used by an information service provider to allow it to deliver more consistent performance to its customers. The *endpoints* of an RDS include a *source node* and a potentially large number of *sink nodes* distributed within a fixed network infrastructure. Sink nodes are typically routers within metropolitan areas where customers of the information service are found. A network provider selects a set of links within the network and dimensions bandwidth reservations on those links in order to accommodate expected traffic flows from the server to the various sink nodes. This allows traffic from the source node to flow through to the sinks without contention from other traffic sources, improving quality of service.

To allow for variability in the traffic volume at sink nodes, reservations are dimensioned based on the mean and variance of the expected traffic. Links that carry large traffic volumes are generally more efficient than links that carry small traffic volumes, since the amount of bandwidth that must be reserved to accommodate traffic variability becomes a smaller fraction of the total as traffic volume grows. This effect makes it beneficial to group together flows going from the source to sinks that are close to one another. An example RDS is shown in Figure 1. Note that as traffic flows diverge to reach different sinks, the total reserved bandwidth on the "downstream links" will generally be larger than the reserved bandwidth on the upstream link (or links).



(a) Reserved delivery network.




(b) Detail view of the bandwidth reservation on a router.

Fig. 1. Reserved Delivery Subnetwork

The problem of configuring an RDS can be formulated as a minimum cost network flow problem [1], in which the cost per unit flow decreases as the flow on an edge increases (this models the declining influence of traffic variability as traffic volume grows). While minimum cost flow problems can be solved efficiently when the cost per unit flow is fixed [2], [3], the problem becomes NP-hard when the cost functions are concave [4]. Current research on such problems centers on enumerative algorithms that can require exponential time in the

worst-case [5], [6] and are not practical for large problem instances. Relatively little work has been done on approximation algorithms.

In this paper, we introduce an approximation algorithm for the RDS configuration problem. The algorithm is a variant of a classical augmenting path algorithm for the minimum cost flow problem with linear costs (constant cost per unit flow). As with the classical algorithm, we seek a minimum cost augmenting path at each step. However, the choice of such a path is complicated by the fact that the relative costs of different paths depend on how much flow is sent along them. We investigate the implications of this problem and devise an approximation algorithm based on one method for resolving the problem. Experimental results show that the proposed algorithm produces results that are generally no more than twice the cost of an easily computed lower bound. We believe this bound to be rather loose and provide evidence that the true performance is significantly better than what is implied by the lower bound.

The rest of this paper is organized as follows: in Section II, we show how RDS configuration can be formulated as a minimum cost flow problem. We present our proposed algorithms in Section III. Experimental results are given in section IV and concluding remarks in Section V.

## II. Problem Formulation

We start with an elementary observation. If the traffic on a link consists of a large number of independent and statistically similar streams, the mean and the variance of the aggregate traffic scales directly with the number of flows. So, we let $\sigma(\mu) = \alpha\mu^{1/2}$ denote the standard deviation of an aggregate traffic flow with mean $\mu$, where $\alpha$ is a parameter. Note that when $\mu = \alpha^2$, $\sigma(\mu) = \mu$. That is, $\alpha^2$ is the mean traffic rate for which the mean and standard deviation are the same. Given a traffic flow with mean $\mu$ and standard deviation $\sigma(\mu)$, a suitable choice for the reserved bandwidth is $\mu + k\sigma(\mu) = \mu + k\alpha\mu^{1/2}$, where $k$ is a small constant (say 3). With these preliminaries, we can now proceed with a formal statement of the RDS configuration problem.

We are given a directed graph (or network) $G = (V, E)$ and two real-valued functions $l(\cdot)$ and $b(\cdot)$ defined on $E$. We refer to $l(e)$ as the *length* of edge $e$ and $b(e)$ as its *bandwidth*. We also define a real-valued *edge capacity* $c(e)$, which represents the mean rate of the largest reservation that can be carried by edge $e$. The edge capacity satisfies the equation $c(e) + k\alpha c^{1/2}(e) = b(e)$ and is equal to $\left(-k\alpha + \sqrt{k^2\alpha^2 + 4b(e)}\right)^2/4$.

We are also given a *source node* $r \in V$ and a set of *sink nodes* $S \subseteq V$, with each sink node $s$ having a mean demand $\mu(s)$. The minimum cost reserved delivery network that satisfies the mean demands, while respecting the capacity limits on the network links can be found by solving a minimum cost flow problem, in which the flow into each sink is given by its mean demand, and the total flow on each link $e$ is bounded by $c(e)$. The cost of a flow $x$ on an edge $e$ is defined to be $l(e)(x + k\alpha x^{1/2})$. The second factor in this expression corresponds to the amount of bandwidth that must be reserved to accommodate a flow of magnitude $x$. Note that the cost function is concave. Given a minimum cost flow that satisfies the demand, the optimal reserved delivery subnetwork is the subgraph of $G$ defined by the edges with non-zero flows. The cost of the subnetwork is the sum of the costs of the flows on its edges.

Note that when there are no limits on edge capacities, the best RDS is always a tree. We expect that in practice, network link capacities will often not be a limiting factor, so that the best RDS may typically be a tree. Even when link capacities are limited, we may wish to constrain the form of the solution so that all traffic going to a single sink is constrained to use the same path, in order to simplify the routing of the traffic (note that in this case, the RDS need not be a tree).

## III. Approximate Minimum Cost Augmentation

One of the classical methods for solving minimum cost flow problems is the minimum cost augmenting path method. This method iteratively selects a *minimum cost augmenting path* from the source to a sink that has unmet demand and adds flow along that path until either the demand has been satisfied or the capacity limit of some edge on the path has been reached. While this method can find an optimal flow when the cost per unit flow on each edge is constant, it cannot be directly applied to the RDS configuration problem, since the relative costs of two different paths can change depending on the magnitude of the flows added to those paths. That is, it may cost less to add $x$ units of flow to a path $p$ than to an alternative path $q$, but it may cost more to add $2x$ units of flow to $p$ than to $q$.

Although we cannot use the minimum cost augmentation algorithm directly in the RDS configuration problem, we can apply similar ideas to construct an approximation algorithm that does not require an enumerative search of the problem space. We start by reviewing some terminology. In the minimum cost maximum flow problem, we seek a flow function $f$ on the edges of the given network. For any node that is not a source or a sink, the sum of the flows on the incoming edges must equal the sum of the flows on the outgoing edges. The flow must satisfy the given capacity constraints on the edges and must satisfy the given demands required by the sinks. Among all such flows, we seek one of minimum cost.

In the minimum cost augmenting path algorithm, at each step we choose an augmenting path from the source to the sink in the *residual graph* for the current flow. For each edge $(u, v)$ in the original graph, the residual graph has an edge $(u, v)$ if $f(u, v)$ is less than the capacity of $(u, v)$ and it has edge $(v, u)$ if $f(u, v)$ is greater than zero. The *residual capacity* of the edge $(u, v)$ is the difference between the capacity and the current flow. The residual capacity of $(v, u)$ equals $f(u, v)$. An augmenting path is just any path in the residual graph from the source to a sink. It is well known [1] that when the cost per

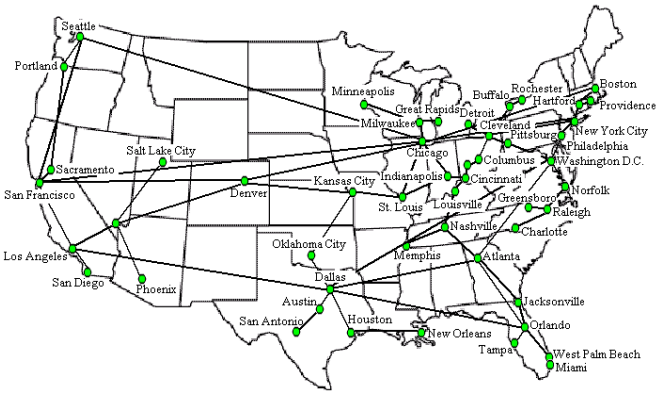



Fig. 2. National network configuration.

unit flow is constant, we can construct a minimum cost flow by finding a succession of minimum cost augmenting paths and *saturating* each one in turn (that is adding as much flow to the path as allowed by the capacity constraints, or the unmet demand at the sink, whichever is smaller).

To apply this approach to the RDS problem, we must first define what we mean by the cost of an edge. For any edge $e$ in the original graph, the cost of carrying $x$ units of flow on $e$ is $l(e)(x + k\alpha x^{1/2})$. We let $\delta_f(e, \Delta)$, be the change in cost caused by adding $\Delta$ units of flow on the edge $e$ in the residual graph, assuming that $\Delta$ is no larger than the residual capacity of $e$. If $\Delta$ is larger than the residual capacity, $\delta_f(e, \Delta)$ is defined to be infinite. We refer to $\delta_f(e, \Delta)$ as the *incremental cost* of the edge $e$, with respect to the increment $\Delta$. The incremental cost of a path, with respect to an increment $\Delta$, is defined as the sum of the incremental costs of its edges. For any flow and increment $\Delta$, we can define a tree $T_f(\Delta)$, which is a shortest path tree rooted at the source in the subgraph of the residual graph defined by the edges with residual capacity no smaller than $\Delta$. The path costs in $T$ are defined with respect to the incremental costs, $\delta_f(e, \Delta)$. As $\Delta$ is increased from zero, we get a finite sequence of trees $T_0, T_1, \ldots, T_m$. For each tree $T_i$ in this sequence, there is a corresponding range $R_i$ of values of $\Delta$.

The *incremental cost per unit flow* of an augmenting path $p$ is $\delta_f(p, \Delta)/\Delta$, where $\Delta$ is the amount of flow needed to saturate $p$. To apply the minimum cost augmentation strategy to the RDS problem, we seek an augmenting path from the source to a sink that has the smallest *incremental cost per unit flow* among all augmenting paths. In principle, this can be done by constructing each of the distinct shortest path trees and selecting the best augmenting path found in all the trees. A computationally simpler alternative is to choose a small set of increments, construct the tree corresponding to each increment, and find the best augmenting path from among this smaller set of trees. While this only "samples" the set of trees, and hence will not always find the best path, it does at least approximate the minimum cost augmentation strategy. We have found that in practice, the best path is usually found in the tree corresponding to the largest increment. This observation has led us to the following simpler algorithm, which we call the *Largest Demand First* (LDF) algorithm.

$f := 0$;
**while** there is unmet demand at some sink
  Let $\Delta$ be the smaller of the largest unmet demand and the largest residual capacity among all augmenting paths.
  Let $p$ be the augmenting path in $T_f(\Delta)$ with the smallest incremental cost per unit flow.
  Modify $f$ by saturating $p$.
**end**

If the algorithm cannot find an augmenting path, while there is still unmet demand, then the algorithm fails. Each iteration of the algorithm requires the computation of a shortest path tree and a *bottleneck shortest path tree*. Both of these computations can be implemented to run in $O(m + n \log n)$ time, where $m$ is the number of edges and $n$ the number of nodes. In networks with ample link capacity, each iteration fully satisfies the demand at some sink, so the number of iterations equals the number of sinks. This leads to an overall running time of $O(s(m + n \log n))$, in the case of ample link capacities. For arbitrary link capacities, the number of iterations can grow exponentially large, as it can for the original minimum cost augmenting path algorithm.

## IV. Experimental Results

To evaluate the LDF algorithm we compared the cost of the solution produced to that of an easily computed lower bound. The lower bound is computed by sorting the sinks in increasing order of their distance from the root and then assuming that each sink is reached by a path of this minimum length, and that the path can be shared with all sinks at greater distances from the root. We evaluated the algorithm on two networks. The first is a $15 \times 15$ torus (each node is connected to four neighbors forming a rectangular grid with "wrap-around edges" linking the top and bottom rows and the leftmost and rightmost columns). Link lengths were uniformly distributed, with the longest links being ten times longer than the shortest. The demands for the sinks were uniformly distributed, all with the same mean demand.

The second network, shown in Figure 2, includes a node at each of the fifty largest metropolitan areas in the United States; the link lengths were chosen to be equal to the geographic distances between the locations, and the demands were chosen to be proportional to the populations of the metropolitan ares. The locations of sources and sinks were selected randomly, with every node having the same probability of selection. For the results reported here, unbounded link capacities were used in both networks. An example RDS computed by the LDF algorithm is shown in Figure 3. The source for this example is

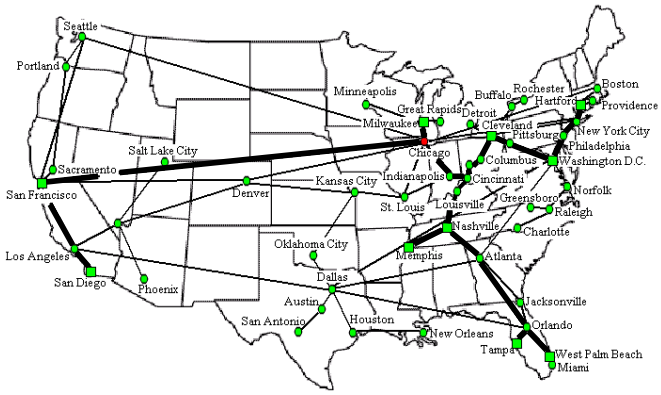
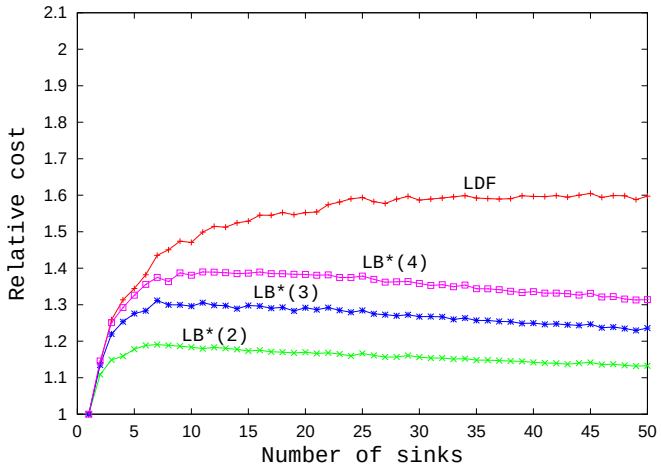



Fig. 3. Example RDS computed by the LDF algorithm



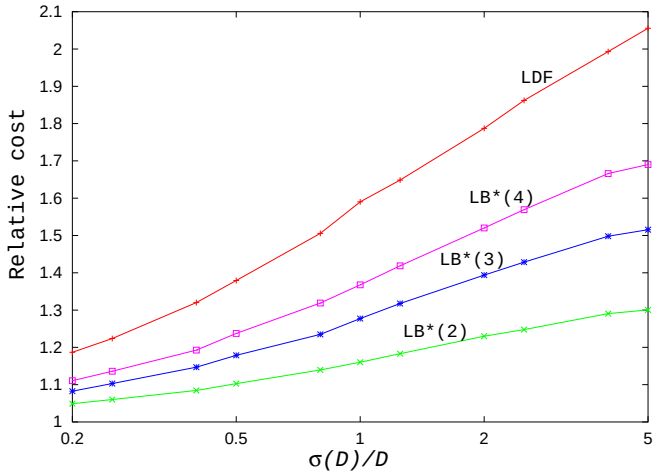




Fig. 4. Performance of LDF on torus network

in Chicago and there are ten sinks at various locations around the country (the sinks are designated by small squares on the map). The cost of this solution is about 1.34 times the cost of the lower bound.

Figure 4 shows how LDF performs on the torus. The first chart shows the ratio of the cost of the solution produced by LDF to the lower bound, as the number of cities increases from 1 to 50, while $\alpha^2$ is fixed so that $\sigma(D) = D$, where $D$ is the average demand per sink. Each data point represents the average of results from 100 independent problem instances. For large numbers of cities, the LDF algorithm produces solutions costing no more than about 1.6 times the lower bound. The curves labeled LB*(2), LB*(3) and LB*(4) are related to the lower bound and provide evidence (although no proof) that for larger numbers of cites the lower bound is fairly loose. LB*(2) is computed by first dividing the sinks into two sets, those to the "left" of the source and those to the "right" of the source. Each of these subsets is then sorted by distance from the source and each node is assumed to share its path to the source with all nodes in the same subset that are at greater distance from the source. LB*(3) (and LB*(4)) is computed similarly, by first dividing the sinks into three (respectively four) sets of nodes defined by "pie-shaped" regions centered on the root, then sorting the subsets by distance from the root and assuming the maximum possible sharing of paths among nodes in the same set. For larger numbers of randomly distributed cities, it's reasonable to expect LB*(2), LB*(3) and LB*(4) to be no larger than the cost of an optimal solution, although they do not constitute true lower bounds. Note that for 50 sinks, LDF produces solutions that average about 1.3 times LB*(3).

The second chart in Figure 4 shows how the performance of LDF varies in comparison to the lower bound as $\alpha^2$ is varied so that $\sigma(D)/D$ varies from .2 to 5, while the number of sinks is fixed at 25. For small values of $\sigma(D)/D$, there is less to be gained from sharing paths, so LDF performs better, relative to the lower bound. For larger values of $\sigma(D)/D$, there is much more to be gained by sharing paths, so the gap between the lower bound and LDF gets larger. When $\sigma(D)$ is five times the average demand per sink, the cost of the solutions produced by LDF increases to about 2.05 times the lower bound.

Figure 5 shows how LDF performs on the national network. We note that LDF performs generally better in this case, than for the torus, but the general character of the results remains the same. We speculate that the improved performance arises largely because the national network spans a greater east-west distance than north-south, and that the large numbers of cities are near the coasts meaning that often the root is near one of the coasts, which makes it relatively easy for LDF to produce solutions with large amounts of sharing. The wide variance in the link lengths in the torus network may also contribute to the reduced performance in that case (some links in the torus network violate the triangle inequality, preventing them from being used in any solution).

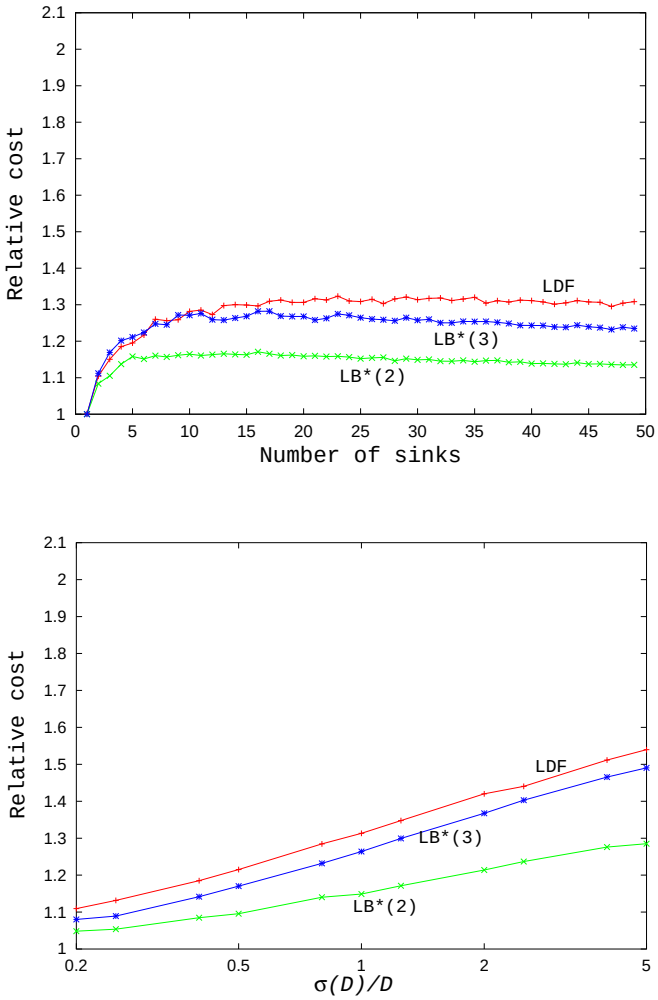



Fig. 5. Performance of LDF on national network

## V. Conclusions

In this paper, we introduced the problem of configuring optimal reserved delivery subnetworks and developed a natural approximation algorithm to solve the problem. Our algorithm is based on the classical minimum cost augmentation algorithm for minimum cost network flows. Our experimental results show that the proposed algorithm works well in both artificial networks and more realistic network configurations. The solutions produced in our experiments never exceed an easily computed lower bound by more than a factor of two, and we provide evidence to indicate that the true performance is significantly better than implied by the comparison to the lower bound.

## References

[1] R. K. Ahuja, T. Magnanti, and J. Orlin, *Network Flows*. Prentice Hall, 1993.

[2] M. Klein, "A primal method for minimal cost flows," *Management Science*, vol. 14, pp. 205–220, 1967.

[3] R. E. Tarjan, *Data Structure and Network Algorithms*, vol. 44. Society for Industrial and Applied Mathematics, 1983.

[4] G. M. Guisewite and P. M. Pardalos, "Minimum concave-cost network flow problems: Applications, complexity, and algorithms," *Annals of Operations Research*, vol. 25, pp. 75–99, 1990.

[5] G. M. Guisewite and P. M. Pardalos, "Algorithms for the single-source uncapacitated minimum concave-cost network flow problem," *Journal of Global Optimization*, vol. 1, pp. 245–265, 1991.

[6] G. M. Guisewite and P. M. Pardalos, "Global search algorithms for minimum concave-cost network flow problems," *Journal of Global Optimization*, vol. 1, pp. 309–330, 1991.