Design Issues of Reserved Delivery Subnetworks

Ruibiao Qiu

In this proposal, we introduce the reserved delivery subnetwork (RDS), a mechanism that can allow information service providers to deliver more consistent service to their customers without perflow resource reservation. In addition to service performance improvements, reserved delivery sub-networks can also provide protection against network resource attacks. Many applications such asweb content delivery services and virtual private networks can benefit from reserved delivery sub-networks. We address a number of issues with the deployment of RDSs. First, we formulate theconfiguration problem of an RDS as a minimum concave cost network flow problem, where the perunit flow cost decreases as the... **Read complete abstract on page 2.**

Design Issues of Reserved Delivery Subnetworks

Ruibiao Qiu

Complete Abstract:

In this proposal, we introduce the reserved delivery subnetwork (RDS), a mechanism that can al-low information service providers to deliver more consistent service to their customers without perflow resource reservation. In addition to service performance improvements, reserved delivery sub-networks can also provide protection against network resource attacks. Many applications such asweb content delivery services and virtual private networks can benefit from reserved delivery sub-networks. We address a number of issues with the deployment of RDSs. First, we formulate theconfiguration problem of an RDS as a minimum concave cost network flow problem, where the perunit flow cost decreases as the current flow increases. An approximation heuristic is presented andstudied to solve this configuration problem. Second, we extend our study to the configuration prob-lem of RDSs with multiple sources. We also investigate the configuration problem for subnetworksthat allow load redistribution and load balancing among the sources. In addition, we plan to studyhow to use RDS proxies to regulate the flow of traffic to end users, so as to minimize network delay.

Design Issues of Reserved Delivery Subnetworks

Ruibiao Qiu

Department of Computer Science and Engineering
Campus Box 1045
Washington University
One Brookings Drive
St. Louis, MO 63130-4899, USA

Abstract

In this proposal, we introduce the reserved delivery subnetwork (RDS), a mechanism that can allow information service providers to deliver more consistent service to their customers without per flow resource reservation. In addition to service performance improvements, reserved delivery subnetworks can also provide protection against network resource attacks. Many applications such as web content delivery services and virtual private networks can benefit from reserved delivery subnetworks. We address a number of issues with the deployment of RDSs. First, we formulate the configuration problem of an RDS as a minimum concave cost network flow problem, where the per unit flow cost decreases as the current flow increases. An approximation heuristic is presented and studied to solve this configuration problem. Second, we extend our study to the configuration problem of RDSs with multiple sources. We also investigate the configuration problem for subnetworks that allow load redistribution and load balancing among the sources. In addition, we plan to study how to use RDS proxies to regulate the flow of traffic to end users, so as to minimize network delay.

Design Issues of Reserved Delivery Subnetworks

Ruibiao Qiu
ruibiao@arl.wustl.edu
Applied Research Laboratory
Department of Computer Science and Engineering
Washington University
St. Louis, MO 63130, USA

1. Introduction

The Internet has become an information infrastructure that we depend on in our daily life. However, the Internet in its current state is not sufficient for mission critical business applications. There is no efficient support for bandwidth reservation mechanisms for flows that require guaranteed bandwidth. Because of the best effort nature of today's Internet, there is no way to distinguish between traffic with transaction-oriented mission critical data and traffic for causal web browsing. These traffic sources have to compete equally for resources such as access bandwidth. Such lack of reserved bandwidth flows makes it difficult for transaction-based business applications to guarantee mission critical operations. Multimedia applications also suffer from such lack of efficient bandwidth reservation because it is hard to maintain quality of service for multimedia flows with the basic best-effort Internet service. In addition, the Internet is vulnerable to malicious attacks, such as denial of service (DOS) and various worm attacks. In January of 2003, the Internet "slammer" worm attack left thousands of bank customers without ATM access, and dozens of flights grounded [1]. The data exchange between the servers at the headquarters of the banks and airline companies and the terminals on ATMs and in airports was severely affected when the Internet got heavily congested. Clearly, the current Internet is an insufficient information infrastructure, and needs great improvements to provide consistent and stable services comparable with traditional information infrastructure, such as telephone networks.

In order to make the Internet a better information infrastructure, various techniques have been proposed to improve services of the Internet. Some solutions, such as active networking, require changes on the routers in the Internet. However, a number of non-technical issues, such as the incurring costs and complicated business agreements among Internet service providers, make the deployment of this class of solutions unrealistic. In contrast, another class of solutions use overlay networks to circumvent the deployment issues. An overlay network is built on top of the regular Internet, and connects hosts involved in data exchange without changing the underlying network operation. New techniques that are not readily available in the Internet can be implemented in overlay networks to improve end-to-end stability, reliability and performance. Overlay networks vary in their architectures. Some overlay networks are composed of special end hosts, or overlay nodes. By extensive exchange of network information among all overlay nodes, they attempt to make intelligent data forwarding decisions [2, 3]. These overlay networks are often used as short-term solutions or vehicles for Internet research because of their inherent limitations. Other overlay networks (such as SON [4]) build on the underlying physical networks with reserved resources exclusively accessible to the users of the overlay networks.

In this proposal, we introduce the concept of a *Reserved Delivery Subnetwork* (RDS). An RDS is designed for information service providers who have the need for delivery of consistent service to their customers even under very extreme network conditions. An RDS provides a subnetwork for an information service provider connecting the provider to its customers at different locations. The traffic source is the server at the information service provider's central location, while the sinks are typically access routers where customers of the information service are found.

The links in the RDS are carefully provisioned with sufficient bandwidth so that traffic from the source node can flow through to the sinks without contention from other traffic sources, improving quality of service. Although it is difficult to provide quality of service for individual flows in the current Internet, RDSs give service providers a way to address the quality of service issue on an aggregate basis. In addition, bandwidth limits on reverse paths provide a protection mechanism against malicious attacks.

Many applications can benefit from the employment of reserved delivery service. One of the most direct applications is web content delivery. A web site or an Internet content provider can purchase such a service from the physical network service provider. With an RDS rooted at the access router where the server resides to all locations where most demands are found, a content provider can deliver consistent service to end users even under extreme network conditions. Another application can be found in enterprise virtual private networks (VPNs) as well as banks and airline companies that depend heavily on the time-critical delivery of transaction-oriented data. In this case, the company headquarters can subscribe to a customized reserved delivery service such that information exchange will not be interrupted even when the network is under attack. It is possible that a service provider and the end users are located in different network domains run by different physical network service providers. Instead of negotiating a multilateral service level agreement with each individual network provider, a special type of service provider can be involved. We can call such service providers as *Reserved Delivery Subnetwork Providers* or *RDSPs*. An RDSP provides reserved delivery service to a customer by constructing an RDS from the customer to their end users. The subnetwork may span multiple network domains. According to the customer requirements, the RDSP purchases reserved bandwidth on subnetwork links from each individual network provider, and gains service revenues from the customers that subscribe to the service from it. In addition, RDSs could extend their applications to grid computing [5], peer-to-peer networks, multimedia networking, and wireless networks.

The rest of the proposal is organized as follows: we first discuss the related work in Section 2. The reserved delivery subnetwork (RDS) architecture is formally introduced in Section 3. In Section 4, we describe the configuration problem for RDS. Two approximation heuristics are presented and compared. In Section 5, we extend our work to deal with RDSs with multiple sources and capable of load redistribution and load balancing. The reserved delivery service not only can provide contention free paths to end users, other performance improvements are also achievable. In Section 6, we present techniques to improve end user application performance by leveraging the underlying RDS. We lay out the research plan for this investigation in Section 7.

2. Related Work

A variety of overlay network solutions have been proposed to improve the end-to-end performance in the Internet. The goal of the Detour routing project [2] is to build an alternative routing infrastructure with an overlay network, as a research vehicle to experiment with performance-sensitive routing algorithms. Similarly in the resilient overlay network (RON) project [3], an overlay network is built connecting a small group of participants for certain distributed applications. The major goal is to implement quick detection and recovery from failures, improving stability and reliability. It takes seconds to detect and recover from a failure, compared with several minutes delay in the ordinary Internet. However, both Detour and RON share some disadvantages. First, a participant node in either a Detour network or a RON constantly exchanges information about the network status with all other nodes in order to make better packet forward decisions. The bandwidth and control overhead limit their applications to small groups of participants (typically no more than 50). Second, the nodes in both a Detour network and a RON are special end hosts. So, traffic flow must traverse the access link of an intermediate node twice when it is forwarded, making the access links potential bottlenecks. In addition, they still rely on the underlying Internet to forward packets with no guarantees. Hence, they only provide short-term solutions.

Another class of overlay network solutions emphasize the resource reservation in the underlying physical networks to provide stable and reliable performance. Duan, Zhang and Hou introduced service overlay networks (SON) in [4]. They intended to address the end-to-end QoS problems by introducing a SON provider between physical network providers and application users. A SON provider purchases bandwidth from multiple network domains with certain guarantees and charges the application users for the end-to-end QoS sensitive service they provide. They studied the

cost recovery problems of SONs, and presented cost models and solutions for static and dynamic bandwidth allocation as well as an online algorithm. In their work, they assume the cost is a linear function of the provisioned bandwidth. Such an assumption is not accurate because it ignores the fact that traffic variation becomes relatively smaller when the volume of traffic grows. A non-linear concave cost function is more appropriate, but makes the problem computationally hard. In addition, they only covered applications requiring point-to-point connections. However, real world application normally involve more than two participants. Simply combining together all point-to-point connections to build a SON may not be efficient.

Recently, research in network provisioning is gaining its popularity because of its effectiveness and ease of deployment. Fraleigh, Tobagi and Diot studied the problem of bandwidth provisioning in backbone networks to support latency sensitive traffic [6]. They modeled the end-to-end delays in backbone networks, and presented a procedure to determine the bandwidth to reduce the latency violations. They found that a small percentage of bandwidth over-provisioning is sufficient for most latency sensitive traffic. Juttner, Szabo and Szentesi compared bandwidth requirements for virtual private networks using the hose model and pipe models in [7]. They also gave a lower bound using the hose model. They concluded in their studies that the hose model is a better model for bandwidth provisioning in virtual private networks. Mitra and Wang used a two-tier market structure to optimize bandwidth provisioning and path selection in networks, where demands are uncertain and specified by probability distributions [8]. Using mean-risk analysis, they were able to find an optimal solution that maximizes the revenue (demands served) while containing the risk (revenue falling below a certain level).

Pappu, Parwatikar, Turner and Wong proposed distributed queueing (DQ) in [9], intended to resolve congestion in a switch fabric caused by sustained overloading at an output port. The distributed queueing algorithm regulates the transmission rates from input ports to output ports to avoid congestion inside the switch fabric. Virtual output queues are maintained at the input port, and the information about the queues at input and output ports is used to determine the transmission rates at the input ports. So, an output port with large backlog gets less traffic from input ports while it drains its backlog, and output ports with small backlog get more traffic. This inspires the traffic flow regulation idea in this proposal. We extend the distributed queueing idea to a larger context in overlay networks, and address similar congestion problems.

3. Reserved Delivery Subnetworks

3.1. Formal Definition

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).
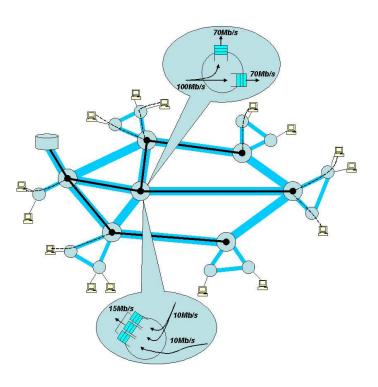
Figure 1: Reserved Delivery Subnetwork.

3.2. Goals and Challenges

Cost Efficiency and Service Effectiveness   The cost of an RDS is crucial its success. If insufficient bandwidth is reserved on some links, demands from users at the location on a path with the "thin" link will not be fulfilled completely. On the other hand, if too much bandwidth is reserved on links, the RDS provider will suffer loss of service revenues. Therefore, an ideal solution would strike a balance between the cost efficiency and service effectiveness.

The problem of configuring an RDS can be formulated as a minimum cost network flow problem [10], 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 [11, 12], the problem becomes NP-hard when the cost functions are concave [13]. Current research on such problems centers on enumerative algorithms that can require exponential time in the worst-case [14, 15] and are not practical for large problem instances. Relatively little work has been done on approximation algorithms.

Fault Tolerance and Recovery   Although in theory we can assume the network providers can provide fault-free networks to the RDS providers, failures could happen in reality. So, the RDS providers should also take such possibilities into account in their RDS configuration.

Improved Application Performance   The reserved delivery service provides guaranteed bandwidth to access routers where the end users are located. In addition to the aggregate bandwidth guarantee, each individual end host can also benefit from the RDS because the RDS provides a relatively stable underlying network. We should be able to leverage such advantages to further improve the end user performance.

4. Configuration of Reserved Delivery Subnetworks

4.1. Problem Formulation

In our previous investigation [16], we introduced an approximation algorithm for the simple RDS configuration problem, where configuration is solely based on the user demand matrix. 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 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.

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 μ, where α is a parameter. Note that when $\mu = \alpha^2$, $\sigma(\mu) = \mu$. That is, α^2 is the mean traffic rate for which the mean and standard deviation are the same. Given a traffic flow with mean μ 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 RDS 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 RDS 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.

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. 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 on which more flow can be added. 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 Δ units of flow on the edge e in the residual graph, assuming that Δ is no larger than the residual capacity of e. If Δ 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 Δ. The incremental cost of a path, with respect to an increment Δ, is defined as the sum of the incremental costs of its edges. For any flow and increment Δ, 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 Δ. The path costs in T are defined with respect to the incremental costs, $\delta_f(e, \Delta)$. As Δ 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 Δ. The *incremental cost per unit flow* of an augmenting path p is $\delta_f(p, \Delta)/\Delta$, where Δ is the amount of flow needed to saturate p.

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).

4.2. Algorithm Design Issues

As we noted previously, the edge cost function is a concave function of the currently carried amount of flow. Thus, when we aggregate more flows on a link, the over-provisioned bandwidth, that is necessary to accommodate traffic variations, decreases, resulting in more cost efficient networks. So, we prefer a configuration algorithm that rewards flow aggregation. However, it is possible that if we favor aggregation to strongly, longer paths may be selected while shorter and cheaper routes exist. Thus, we need also to restrict the path selection within a reasonable region.

When we select a path from the root to a sink, we can either keep all traffic to the sink on a single path, or split it among a number of paths leading to the sink, some of which may not have enough capacity for the sink by themselves. The concave edge cost function suggests that keeping the traffic flows together is more cost efficient than splitting them. However, such a strategy is not always able to satisfy all sinks in networks with limited link capacities, which leads to higher demand blocking ratio (the ratio of unmet demands to the total demands) than an algorithm that splits flows. Therefore, when we design an RDS configuration algorithm, we need to consider the tradeoff of flow splitting and aggregation, and try to reduce the cost while minimizing the possibility of sink blocking.

4.3. Path Augmentation Algorithm

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. 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. It is well known [10] that when the cost per 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 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. There are various strategies to select the set of increments. Because our goal is to schedule flows to the sinks, we should select increments related to the sink demands. In order to make such a selection, we can order the sinks in a specific order, and use the remaining unmet demand as the increment values (Δ). If a path is found within $T_f(\Delta)$, we can then augment the flow along the path.

Note that the flow augmentation can also be implemented with various strategies, resulting in RDSs with different costs. The following pseudo code shows the generic framework of our algorithm. Depending on the sink sorting and path augmentation strategies, different algorithms can be obtained.

Order the sinks $s_1, \cdots, s_m$ according to a certain sorting strategy
for $i \in [1, m]$
    Augment flow to satisfy demand to s_i with a certain augmentation strategy
end

Each iteration of the algorithm requires the computation of a shortest path tree and possibly 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.

4.3.1. Sink Ordering Strategies

Largest Demand First (LDF) The *Largest Demand First* (LDF) algorithm orders the sinks by their demands such that for sink $s_i \in \{s_1, s_2, \cdots, s_m\}$, $\mu_i \geq \mu_{i+1}$. LDF establishes paths to the sinks with the largest demands first. Therefore, the flows on existing paths are large, and the costs benefits of sharing a path to the root by subsequent sinks are high. 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. However, in networks with limited capacity, it is possible that some sink demands will not be satisfied after the same number of iterations.

Shortest Distance First (SDF) The Shortest Distance First (SDF) algorithm is another sorting algorithm which orders the sinks by their physical shortest path distance to the root. That is, for the sorted sinks $\{s_1, s_2, \cdots, s_m\}$, $l_i \leq l_{i+1}$ for $1 \leq i < m$ where l_i is the length of the physical shortest path p from s_i to the root. SDF tries to take advantage of sinks that are close together. By ordering sinks by their distances to the root, sinks that are close to each other are likely to share the same path to the root, resulting in a more cost efficient network.

Maximum Demand/Flow First (MDF) From the cost function, we can see that the cost increment incurred by adding Δx units of flow on edge e is $l(e)[\Delta x + k\alpha(\Delta x + x_0)^{1/2} - k\alpha x_0^{1/2}]$, where x_0 is the original amount of flow on e. Thus, the incremental cost is not only affected by the edge length and flow increment, but also by the current flow on the edge. Consequently, the cost increment is also affected by the current flow on paths because the cost increment on the path is simply the sum of the edge cost increments of all the edges in the path. The Maximum Demand/Flow First (MDF) attempts to include the current flow factors in ordering the sinks.

In order to factor in the current flow, we note that the incremental cost (with regard to the flow increment Δx) can be written as

$$l(e)[\Delta x + k\alpha(\Delta x + x_0)^{1/2} - k\alpha x_0^{1/2}] \approx l(e)\{[1 + k\alpha/(2x_0^{1/2})]\Delta x - k\alpha/(8x_0^{3/2})\Delta x^2\}$$

and the unit incremental cost then is approximately $l(e)[1 + k\alpha/(2x_0^{1/2}) - k\alpha\Delta x/(8x_0^{3/2})]$. We thus introduce a new metric

$$f = \sum_{e \in p} l(e)[1 + k\alpha/(2x_0^{1/2}) - k\alpha\Delta x/(8x_0^{3/2})]$$

where p is the augmenting path, e is an edge on p, and Δx and x_0 are the flow increment and current flow on e, respectively. We can use f as a factor when we order the sinks. In particular, we first order sinks by their demands in a decreasing order. Sinks are divided into subsets, and each subset holds sinks with similar demands. Within each subset, create $T_f(b)$ for each sink, where b is the smaller of the unmet demand of that sink and the largest residual capacity among all paths to that sink. Compute f for all T_f, and then sort all sinks by the results.

Sink Sorting	Path Augmentation
LDF	SFA
SDF	MCA
MDF	HA
CDF	DCA
	$(DCA_\gamma, DCA_\beta, DCA_\epsilon)$

Table 1: Feasible algorithms.

Combined Distance/Flow (CDF)   The Combined Distance/Flow (CDF) algorithm uses a similar idea to MDF. Instead of first ordering sinks into subsets by their demands in a decreasing order as in MDF, CDF orders sinks by their physical distances in an increasing order first. Subsets of sinks with similar physical distance are first formed, and $T_f(b_i)$ is created for each sink i within a subset, where b_i is the smaller of the unmet demand of that sink and the largest residual capacity among all paths to that sink. Sinks are then ordered by their metrics f.

4.3.2. Path Augmentation Strategies

Single Flow Augmentation (SFA)   The Single Flow Augmentation (SFA) algorithm always tries to augment a flow to a sink in a single path. If no such path can be found while there is still unmet demand, then the algorithm fails. 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 still equals the number of sinks, but there are sinks whose demands cannot be satisfied, resulting in blocking situations. In addition, SFA results in lower cost network when the link capacity is not a limiting factor because it avoid the "penalty" of splitting flows. The obvious drawback is blocking in more congested networks.

Most Cost-efficient Augmentation (MCA)   The Most Cost-efficient Augmentation (MCA) is the opposite to SFA. It always tries to augment along the paths that give the minimum per unit incremental cost. SPA has the benefits of low blocking as long as there is residual capacity to the sink. However, it may create more fragments of flows, which increases the cost of the resulting network.

Hybrid Augmentation (HA)   The Hybrid Augmentation (HA) algorithm combines the previous algorithms by first attempting to add a single flow and reverting to the most cost-efficient path when there is no single path with enough capacity.

Distance Constrained Augmentation (DCA)   The Distance Constrained Augmentation (DCA) algorithm constrains the selection of augmenting paths to a sink by the physical shortest distances to the root from the sink. The goal of this algorithm is to avoid unnecessarily long paths. γ is a pre-defined parameter. We refer to this algorithm as Fixed Range DCA, or DCA_γ due to the parameter γ. Another DCA implementation adds more flexibility by using varied ranges. We refer to this DCA implementation as the Varied Range DCA, or DCA_β due to the use of parameter β. A more extensive but time-consuming DCA implementation method checks a range of parameters to determine the optimal results. We refer to this DCA implementation method as the Best Range DCA, or DCA_ϵ due to its use of parameter ϵ.
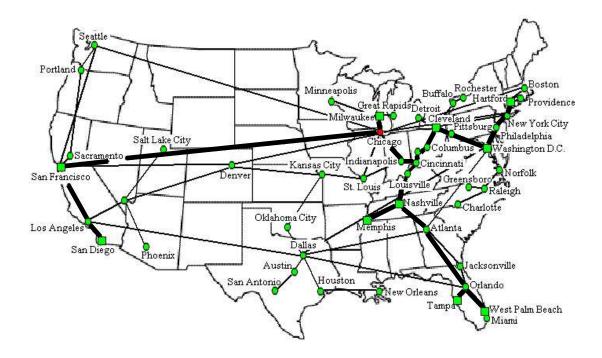
Figure 2: An example RDS in a national network topology.

4.3.3. Comparison of Different Path Augmentation Algorithms   Depending on the choice of different strategies in the sorting and augmentation steps, we can obtain different algorithms as listed in Table 1. There are 24 feasible algorithms with the listed sink sorting and path augmentation strategies using the our solution scheme. In our simulation, we refer to an algorithm in the form of "S-A", where "S" is a sink sorting strategy, and "A" is an augmentation strategy.

4.4. Evaluation

In order to evaluate the different algorithms under our approximation heuristic, we simulate the algorithms on network topologies with variable sizes and demand variations. In our previous study, we examined the performance of the LDF algorithm on a torus topology and a network that includes the fifty major metropolitan areas in the United States [16]. Fig. 2 shows (in highlight links) an example RDS in a national network with fifty major metropolitan areas. In this example RDS, Chicago is the root node, and there are ten sinks at various locations around the country.

Fig. 3 shows how the LDF algorithm performs on the national network. The first chart shows the ratio of the cost of the solution produced by LDF to a lower bound, as the number of sinks increases from 1 to 50, where α^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
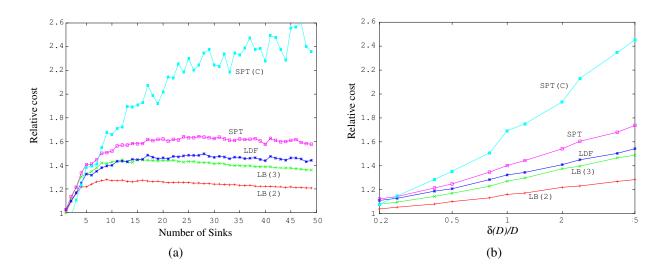
Figure 3: Cost comparison of algorithms in national network topologies.

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. We also include two curves labeled "SPT" and "SPT(C)" for comparison. The "SPT" curve corresponds to the traffic delivery cost using a shortest path tree network topology, and the "SPT(C)" curve is the cost of a star network assuming a complete connected topology of all metropolitan areas. Note that for 50 sinks, LDF produces solutions that average about 1.1 times LB*(3). The second chart in Fig. 3 shows how the performance of LDF varies in comparison to the lower bound as α^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. The charts also show that simply scheduling traffic flows along the shortest paths would not results suboptimal costs because there is less degree of traffic sharing. As in Fig 3a, a shortest path network has an average cost about 1.2 times of the LDF solutions, while the star network in the complete network as the average cost more than twice of the LDF solutions. When $\sigma(D)$ is five times the average demand per sink, the cost of the solutions produced by LDF increases to about 1.55 times the lower bound.

To expand the evaluation sample space and avoid biases towards certain topologies, we plan to conduct more performance studies in other cases to develop greater insight into the performance of the algorithms.

5. Reserved Delivery Subnetworks with Multiple Sources

In the previous section, we investigated RDSs with a single source. In this section, we consider RDSs with multiple sources. These RDSs come into play when we want to improve performance and reliability. Using multiple distributed servers allows lower delay and better service. In addition, we would allow some load redistribution and load balancing to handle irregular traffic and failures, so that the subnetworks are more reliable.

5.1. Source Replica Placement

Delivering data from a single source node to all sinks could lead to long delay for sinks far away and therefore long response time and performance. It could also be a single point of failure, decreasing the reliability in case of link or node failures. Therefore, replicating source data onto multiple servers can improve both performance and reliability.
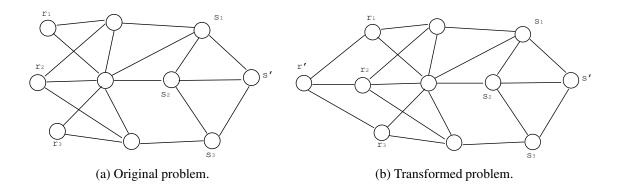
(a) Original problem. (b) Transformed problem.

Figure 4: RDS with multiple sources.

Shi and Turner [17] studied the server placement problem in overlay networks, and used a set cover algorithm to find solutions. Qiu, Padmanabham and Voelker [18] investigated how to place web server replicas cost efficiently, and found that a greedy algorithm based on distances and requests performed best for trace-driven tests. Although their proposed methods are good starting points, placement of multiple server replicas in an RDS requires some different methods. This is mainly because previous work assumes linear edge costs, while the cost is concave on edges in an RDS.

To investigate the source replica placement problem, we will start with the greedy algorithm in [18] and the set cover algorithm in [17], and observe the effects of concave edge costs on the performance of the algorithms. Besides, the bandwidth for each source must be allocated according to the sinks it serves.

5.2. Simple Configuration Problem with Multiple Sources

This section extends our prior work to handle subnetworks with multiple sources. The configuration problem of an RDS can be transformed into a corresponding single source RDS configuration problem. Assume the source nodes are $R = \{r_1, r_2, \ldots, r_n\}$. We can create a pseudo source node r', and connect r' to r, where $r \in R$. Make the capacity on the newly added link (r', r) equal to the total bandwidth B_r of the source node r, and the length of (r', r) be 0. By this transformation, an RDS with multiple source nodes becomes an RDS with r' as the single source node, and $\sum_{r \in R} B_r$ as the total source capacity. Fig. 4 shows an example of such transform. After this transformation, we can apply the algorithm for single source RDS configuration directly.

5.3. Dynamic Load Redistribution

In this section, we want to investigate the configuration problems for RDSs with multiple sources in which extra capacity is set aside to allow for load redistribution among sources. This is needed when there is a dramatic increase of demands in some sinks such that the original source can no longer handle the user demands. It is also helpful to quickly recover when a host or link failure make a source inaccessible.

In order to study the problem, we first introduce the concept of "coverage" of a source node. The coverage of a source node r is a set of nodes, which includes sink nodes that receive traffic from r as well as nodes on the paths from r to these sink nodes. It is clear that no two sources has the same coverage, and the union of all source coverage is the set of all nodes in the subnetwork. In addition, denote the extra bandwidth as B_e, and bandwidth of source r as B_r. We refer to a source node that experiences dramatic traffic demand increase as a "deficit source" and the incremental traffic demand as "deficit". Thus, we can handle deficit up to B_e with extra set aside bandwidth B_e. We also refer to a source node that can serve the traffic demand for a deficit source as a "substitute source". Therefore, in case of a deficit at a source node, its substitute sources will take over traffic demands from a subset of its sinks .
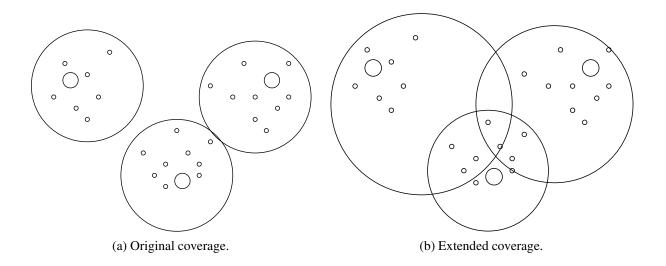
(a) Original coverage.

(b) Extended coverage.

Figure 5: Load redistribution.

In order to enable dynamic load redistribution, we need to decide a set of deputy sources for each possible deficit source, and allocate the extra bandwidth among the deputy sources such that a maximum number of sinks originally served by the deficit node can be served with the least cost. A simple solution to determine these sets is to start with the subnetwork computed previously without load redistribution. The coverage set C_r is constructed for each source node r. For each source node r, we pick the source node r_d that is the closest to nodes in C_r as the deputy source node, and recompute the RDS with $B_{r_d} + B_e$ as the bandwidth of r_d. In the new subnetwork, r_d could cover more sinks. Record the "extended" coverage of r_d and the overlapped sinks nodes. Next, we pick two closest sources as possible deputy sources, and divide B_e between them, either equally or proportionally to the distance of the sinks close by. Then, a new RDS is computed with the updated source bandwidth, and the new extended coverage and overlapped sinks are recorded. Repeat the above procedure in a similar way until the possible deputy sources include all other sources. Compare all the coverage, overlapped sinks, and cost increments, the set of sources that has the largest overlapped sinks and the least cost is chosen as the deputy sources for r. We repeat this procedure for all sources to determine the deputy sources. Fig. 5 shows an example of extended coverage.

6. Traffic Regulation in Reserved Delivery Subnetworks

Clearly, the exclusive bandwidth access in an RDS can improve the end-to-end performance. Besides the benefits of exclusive bandwidth access, we argue that there is potential for further end-to-end performance improvements in an RDS. In this section, we present techniques to improve end-to-end performance in an RDS by regulating traffic flows from the source to sinks.

6.1. Regulating Traffic Flows in an RDS

In an RDS, when a sink node is under sustained overload, and the demand is kept on a higher than average level, flows to such a sink will over-consume more bandwidth resources than it should be allocated on upstream links in the path from the root to the sink. This resource over-consumption will lead to a undesirable situation where the other sinks sharing some of the same upstream links to the root receive less bandwidth resources than their demands, even though the links close to the these sink nodes have sufficient bandwidth. As a result, the reserved bandwidth on these links is under utilized, and the users at the locations of these affected sinks will experience reduced quality of service.
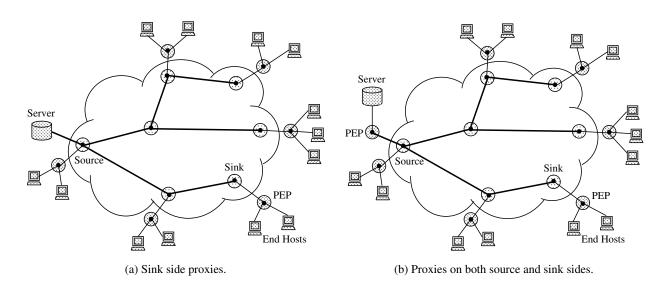
(a) Sink side proxies.

(b) Proxies on both source and sink sides.

Figure 6: Architecture alternatives.

This unbalanced resource utilization on RDS links and reduced bandwidth on affected sinks is similar to the blocking problem in a packet switch with a sustained overloaded output port. By applying similar ideas from the distributed queueing algorithm for packet switches [9], such problems can be resolved in an RDS, and end-to-end performance improvements can be achieved. The essential idea is to regulate the transmission rates of the source node to sink nodes according to the backlog to individual sink nodes, such that bandwidth utilization on links is balanced and maintained at high level, and the quality of service at a sink node is not affected by other overloaded sinks.

6.2. Performance Enhancing Proxies

Recently, proxy servers see increasing deployment and acceptance in the Internet as a viable means for performance improvements for end users. A special class of proxies that are often employed to improve the performance of TCP are gaining their popularity. They are referred to as performance enhancing proxies (PEPs).

The use of PEPs in an RDS can be justified when we compare it with the traditional end-to-end TCP connections. The throughput of a connection is decided by the end-to-end latency and bandwidth. Assume an end host connects to a sink node with 500 kbps, with a nominal round trip delay (without queueing) of 80 ms. Suppose we insert a proxy that splits the end-to-end connection into two segments, where the connection from the source to the proxy has a bandwidth of 1Mbps, and a round trip delay of 70 ms. The round trip delay between the proxy and the end host is roughly 10 ms. If we want to transmit a document of 10KB, the effective throughput with end-to-end connection is about 330 kbps, while the connection through the proxy gets an effective throughput of 470 kbps. In addition, the informed transport protocol and traffic flow regulation can use aggregate traffic flow information that can only be obtained from a proxy.

As Border et al. summarized in [19] PEPs generally employ a number of mechanisms to achieve their goal of performance enhancement: special ACK handling, tunneling, compression, special handling of periods of link disconnection with TCP, priority-based multiplexing, and "protocol boosters" [20]. The PEPs could be located on the sink nodes of an RDS, or on both the source node and sink nodes. If they are on both sides, the communication between the the PEPs could use standard protocols such as TCP, or some special protocols. The sink side only architecture is illustrated in Fig. 6(a), and the architecture with PEPs on both the source and sink sides is shown in Fig 6(b).

In contrast to end host modification approaches, the performance enhancing proxies approaches provide a richer set of options for performance improvements, and thus have greater potential for performance improvements. This is because proxies located close to the access routers of end hosts, are able to gather aggregate information about the
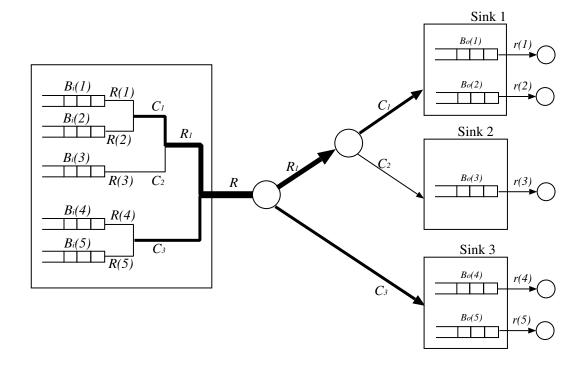
Figure 7: Per-connection traffic flow regulation.

underlying overlay network such as the status of end-to-end paths. In this section, we present the source traffic control and aggregate flow monitoring on performance enhancing proxies to improve end-to-end performance.

6.3. Regulating Traffic Flows

6.3.1. Per-connection Traffic Regulation   A transport protocol such as TCP keeps information about an individual connection on hosts at both ends of the connection (for example, the Internet PCB and TCP PCB in BSD implementations). In order to implement traffic flow regulation, we need to include additional information for each connection. Specifically, we need to keep track of how fast the receiver is consuming data from the sender, how much data is waiting to be forwarded to the receiver at the sender, and how much data is to be transmitted to the receiver at the sink side proxy. For each connection from the sender to an end host x, we define the input backlog $B_i(x)$ as the amount of data that the sender has to send to x, the output backlog $B_o(x)$ as the data backlog awaiting delivery to the receiver on the sink node, and the drain rate $r(x)$ as the rate at which the receiver receives data from the sink node. The drain rate and output backlog for a connection can be constantly measured at the sink side proxy that monitors the flows to end hosts. This information is then fed back to the sender. At the sender side, a *virtual sink queue* (VSQ) is maintained for each connection. A VSQ is the counterpart of a *virtual output queue* (VOQ) used in source routers [9], and keeps tracks of data to be forwarded to a specific end host at the other end of the connection. This additional per-connection information does not consume too much system resources.

The sender allocates transmission bandwidth within the total reserved bandwidth constraints for each connection based on the input backlog, the output backlog, and the receiver drain rate. To enforce the reserved bandwidth constraint on all end-to-end paths, the reserved bandwidth on all links within the overlay network is kept at the sender consistent with the RDS topology. These bandwidth constraints are checked when the sender determines the allocation of transmission rate for each end-to-end connection.

Fig. 7 shows an example per-connection traffic flow regulation architecture. In this example, there are three sink side proxies, and five end-to-end connections to end hosts a, b, c, d and e. The total reserved bandwidth that is allocated for the source is R, and the reserved bandwidth on the other links are R_1, C_3, C_1, and C_2.

$$R_1 \geq C_1 + C_2, \quad R \geq R_1 + C_3 \tag{1}$$

Assume the sender assigns a transmission rate $R(x)$ for receiver x. These transmission rates are subject to the reserved bandwidth constraints:

$$R(1) + R(2) \leq C_1, \quad R(3) \leq C_2, \quad R(4) + R(5) \leq C_3 \tag{2}$$

If (1) and (2) are all enforced, internal blocking can be avoided.

If we denote T as the scheduling period (the interval between two updates of the backlog information), the sender can allocate bandwidth using the following algorithm:

For each sink x, let $t(x) = B_0(x)/r(x)$
Sort the sinks in the increasing order of $t(x)$
For each sink x (in order)
    Let $R(x) = min\{B_i(x)/T, R_a\}$,
        where R_a is the maximum rate allowed by the previously allocated rates and rate constraints
end

However, this basic algorithm relies on timely and precise measurement of $B_i(x)$, $B_o(x)$, and $r(x)$. Unfortunately, such measurements are usually unavailable, especially in the wide area environment because of latency and feedback control overhead. Thus, it only provides coarse grained control, and its effectiveness is affected by the round trip latency on end-to-end paths. In this algorithm, we assume that the estimated end host drain rates are sent back to the sender in an update interval of T. It attempts to clear the data backlogs that can be drained quickly first by sorting the connections by the estimated drain time in an increasing order. For a flow with the shortest estimated drain time, compare its input backlog with the traffic allowed by the reserved bandwidth to the destination sink node. If the input backlog is greater, then send data to use up the reserved bandwidth. If there is remaining bandwidth after cleaning out the input backlog, then allocate the residual bandwidth to the other flows, using the flow with the shortest estimated drain time first. Repeat the above procedure until all input backlogs are cleared, or all reserved bandwidth is consumed.

6.3.2. Aggregate Traffic Regulation   If there is a large number of flows in an RDS, the additional per-flow information could be overwhelming and the per-flow traffic flow regulation could limit the performance. In this section, we propose an aggregate traffic flow regulation algorithm to handle large numbers of connections.

Fig. 8 shows a simplified diagram of aggregate traffic flow regulation. In contrast to the per-connection traffic flow regulation algorithm, a VSQ is maintained for each sink node instead of each flow, and each sink node only maintains an aggregate queue for *all* end hosts connected to the sink node instead of one queue for each end host. The sink node measures the aggregate drain rate, and all connections to the same sink node share the same input backlog, output backlog and drain rate.

The per-connection traffic flow regulation algorithm introduced previously can be modified to use aggregate traffic flow regulation. For the basic algorithm, first check if the total drain rate is higher than the reserved bandwidth. If not, assign drain rate as the transmission rate to the sink. Otherwise, allocate bandwidth to a sink node according to the computed urgency parameter for each sink node within the reserved bandwidth constraint. The urgency parameter has similar definition except that it is for each sink node instead of end host.

In the second aggregate traffic flow regulation algorithm, we first sort the aggregate flows by their estimated drain time at sink nodes in an increasing order. Then, pick the VSQ with the shortest estimated drain time, and check if it is sufficient to drain the input backlog. If not, transmit as much as allowed by the reserved bandwidth. Otherwise, clear out the input backlog, and use the remaining bandwidth for the other VSQs.
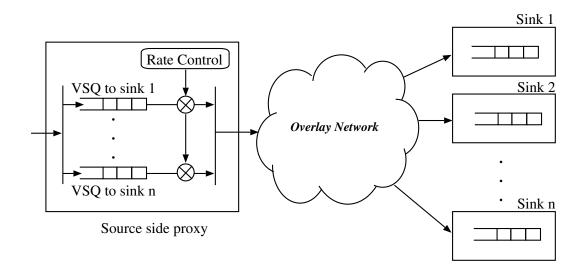
Figure 8: Aggregated traffic flow regulation.

(a) Basic multiple traffic flow regulation.

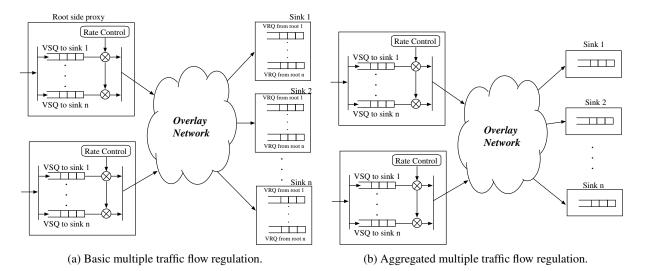(b) Aggregated multiple traffic flow regulation.

Figure 9: Multiple traffic flow regulation.

It should be noted that such rate regulation is coarse grained because rates are determined based on the past information that lags at least one round trip delay. The frequency of the control messages and rate adjustment should be carefully chosen to reduce the overhead and achieve effective rate control.

6.3.3. Multiple Source Regulation   The above traffic flow regulation algorithms are for overlay networks with a single data source. In an RDS, if there is more than one data source, the traffic from different sources could compete for link bandwidth. It is clear that as the number of source nodes increases, more nodes are likely to be affected by overloaded sink nodes. Note, if the reserved bandwidth is exclusively for an individual source node, the blocking caused by an overloaded sink node will be limited to flows from the same source node. In this case, the single traffic flow regulation is sufficient.

However, in overlay networks with multiple data sources with shared reserved bandwidth among different source nodes, more complicated traffic flow regulation algorithms should be used to account for the additional senders. The major challenge is to coordinate the transmission rates from different source nodes to the same sink node without causing internal blocking. As shown in Fig. 9a, there is more than one data source node sending to a set of sink nodes. One sender has to consider other source nodes to the same sink when allocating its transmission rates to sink nodes. In particular, a sender with a larger input backlog to a sink node should get higher transmission rate than the ones with smaller input backlogs. The rate allocation should also consider the output backlog and drain rates as for the single source case. Similarly, the reserved bandwidth limits at both source and sink nodes should be observed.

We need to modify our previous definitions to handle multiple source nodes. We denote m as the number of source nodes, $B_i(i, j)$ as the input backlog from source i to end host j, $R(i, j)$ as the transmission rate from source i to end host j, and R_i as the reserved bandwidth at source node i. One tricky thing about the multiple traffic flow regulation is how a source node can get information about input backlog of other source nodes. One solution is to let the sink node proxy collect it from source nodes. Thus, a source node not only receives information from sink nodes, but also sends out its input backlog to all sink nodes. The sink node gathers and sends back input backlog from all source nodes to a specific end host to all source nodes that send it. In this algorithm, the rates are first allocated by the input backlog and the bandwidth limit at sink nodes. Then, they are checked against the source side bandwidth limits, and are scaled down if necessary.

In the basic multiple traffic flow regulation algorithm, because a source node needs to multicast its input backlog to all sink nodes, the control message overhead grows dramatically, making it less effective and less efficient. In this case, an aggregate multiple traffic flow regulation algorithm is more favorable as it reduces the control overhead. Sources send their backlogs to sinks which aggregate and send sources the aggregate input backlog information. Sources can use this to limit their use of shared resources to prevent overuse. Fig. 9b shows an overlay network with aggregate multiple traffic flow regulation. As in the single source case, a single queue is maintained for all flows from a sink node to all end hosts connected, and only aggregate drain rates are fed back to the source nodes.

In this algorithm, $B_i(i, j)$ is the input backlog from source i to sink j, and $R(i, j)$ is the transmission rate from source i to sink j.

6.3.4. Work-conservation Study of Traffic Flow Regulation   In a switch, a scheduling algorithm is *work-conserving* if for all traffic patterns and all time steps, cells are forwarded on every link for which cells are presented. We can further extend this concept to traffic flow regulation. That is, a traffic flow regulation algorithm is work-conserving if it can forward to a sink whenever there is data destinate for that sink. Most scheduling algorithms in switching systems require a speedup inside the switching fabric to be work-conserving. We will study the work-conservation issue for traffic flow regulation in RDSs. In particular, we will determine if speedup is needed in an RDS to achieve work-conservation? If so, what are the deciding factors of the speedup required for work-conservation? The answers to these questions can help us to identify a good traffic flow regulation algorithm and determine the required bandwidth requirements.

7. Research Plan

7.1. Configuration of Single Source RDSs

We plan to study the performance of the bandwidth provisioning algorithms described in Section 4 through analysis and simulation. Currently, we have some preliminary results indicating that the Distance Constrained algorithms give better performance than other alternatives. However, we plan to conduct more extensive simulation studies with various network topologies, traffic mixes, and demand models. The simulation tool is currently a customized tool. In order to better simulate larger networks in operation, we also plan to conduct simulation with the network simulator (ns-2) [21].

We plan to use more network topologies to test our algorithms thoroughly. The network topologies we consider include radial grids and random donuts. A radial grid is a network consisting of multiple layers of circles with the same center. The source node is at the center, and the sink nodes are evenly distributed on the outermost circle. Intermediate nodes evenly distribute on the circles, and evenly spaced links connect the neighboring circles. If we fix the diameter of the outermost circle, and vary the number of intermediate circles and nodes, we can control the density of the resulting network. A random donut is a network with two circles with a common center. Source node is at the center of the circles, while sink nodes and intermediate nodes are uniformly randomly distributed in the space between the inner and outer circle. When the inner circle shrinks to the center node, it becomes a random disk topology. We will also compare the proposed algorithms with the results obtained from search-based heuristics for small size networks.

7.2. Configuration of Multi-source RDSs with Load Balancing

Similarly, in order to evaluate the configuration algorithms of multi-source RDSs with load balancing capability, we plan to study the performance of the algorithms with analysis and simulation. We first plan to study the problem of cost efficient placement of multiple root nodes in networks. This study will be carried out through simulation of various placement algorithms on different network topologies. Traditionally, server placement algorithms consider latency as the only metric. We would focus on the total costs of the results, and compare the performance of different algorithms. In some cases, it is equivalent to the latency metric, but it is not necessarily always the case. The second part of the study concerns the performance of algorithms for the simple configuration problem with multiple sources. This will be conducted through simulation and analysis. We expect the performance results to be similar to those as the single source cases because it requires a simple transformation between the multiple source cases and the single source case. To investigate the performance of configuration algorithms for multiple source networks with dynamic load redistribution, we plan to simulate situations when one or multiple source nodes become unavailable, and study how different algorithms will respond to these situations by measuring how effectively the load is redistributed to the other active sources and how will the total costs are affected.

7.3. Traffic Regulation in an RDS

The evaluation of traffic regulation in an RDS will be conducted through simulation and implementation. In the simulation studies, we plan to modify our current simulation tool to introduce performance enhancing proxies with the capability of regulating traffic flows from source nodes to sink nodes. An ns-2 object implementing the same functions will be implemented for the ns-2 simulator. We plan to measure the throughput at the source and sink nodes as well as the bandwidth utilization on the RDS links. We plan to test with different network topologies and sinks demand distributions. For the implementation part of our studies, we plan to implement a performance enhancing proxy with traffic flow regulation capability. It is natural to implement the performance enhancing proxy on the access routers at the locations with end host demands. We could explore the possibility of implementation on a programmable hardware platform [22]. However, the memory required may be prohibitively high, especially at the sink node side, making it infeasible to fit in a port. We plan first to study resource (memory and processing power) requirements for a performance enhancing proxy through analysis as well as simulation. It is foreseeable that a performance enhancing proxy implemented on a router platform is able to handle a moderate number of sinks.

In order to overcome the memory limitations on access routers when the number of sink nodes is large, a performance enhancing proxy can be implemented as a stand-alone entity. The popular open source Squid web cache proxy system [23] available on most Unix style platforms can be used as a starting point. Codes of traffic flow regulation can be integrated into the Squid system. We plan to use NetBSD as our platform to implement the modified Squid system with traffic flow regulation capability.

References

[1] CNN, "Computer worm grounds flights, blocks ATMs." URL http://www.cnn.com/2003/TECH/internet/01/25/internet.attack/.

[2] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan, "Detour: a Case for Informed Internet Routing and Transport," *IEEE Micro*, vol. 19, pp. 50–59, January 1999.

[3] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proceedings of 18th ACM SOSP*, (Banff, Canada), October 2001.

[4] Z. Duan, Z.-L. Zhang, and Y. T. Hou, "Service Overlay Networks: SLAs, QoS and Bandwidth Provisioning," in *Proceedings of 10th IEEE International Conference on Network Protocols (ICNP)*, (Paris, France), Novmber 2002.

[5] A. Kothari, S. Suri, and Y. Zhou, "Bandwidth constrained allocation in grid computing," in *Proceedings of Workshop on Algorithms and Data Structures (WADS'03)*, (Ottawa, Canada), July 2003.

[6] C. Fraleigh, F. Tobagi, and C. Diot, "Provisioning IP Backbone Networks to Support Latency Sensitive Traffic," in *Proceedings of IEEE InfoComm*, (San Francisco, CA, USA), April 2003.

[7] A. Jttner, I. Szab, and ron Szentesi, "On Bandwidth Efficiency of the Hose Resource Management Model in Virtual Private Networks," in *Proceedings of IEEE InfoComm*, (San Francisco, CA, USA), April 2003.

[8] D. Mitra and Q. Wang, "Stochastic Traffic Engineering, with Applications to Network Revenue Management," in *Proceedings of IEEE InfoComm*, (San Francisco, CA, USA), April 2003.

[9] P. Pappu, J. Parwatikar, J. Turner, and K. Wong, "Distributed queueing in scalable high performance routers," in *Proceeding of IEEE Infocom*, (San Francisco, CA, USA), April 2003.

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

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

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

[13] 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.

[14] 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.

[15] 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.

[16] R. Qiu and J. S. Turner, "Configuration of reserved delivery subnetworks," in *Proceedings of IEEE Globecom*, (Taipei, Taiwan), Novmber 2002.

[17] S. Shi and J. S. Turner, "Placing servers in overlay networks," in *Proc. Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, (San Diego, CA, USA), July 2002.

[18] L. Qiu, V. Padmanabham, and G. Voelker, "On the placement of web server replicas," in *Proceedings of IEEE INFOCOM 2001*, (Anchorage, AK, USA), April 2001.

[19] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "RFC 3135: Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations." RFC 3135, June 2001.

[20] D. Feldmeier, A. McAuley, J. Smith, D. Bakin, W. Marcus, and T. Raleigh, "Protocol boosters," *IEEE Journal on Selected Areas of Communication*, vol. 16, April 1998.

[21] VINT, "Network Simulator." URL http://www.isi.edu/nsnam/ns/.

[22] S. Choi, J. Dehart, R. Keller, F. Kuhns, J. Lockwood, P. Pappu, J. Parwatikar, W. D. Richard, E. Spitznagel, D. Taylor, J. Turner, and K. Wong, "Design of a high performance dynamically extensible router," in *Proceeding of DARPA Active Networks Conference and Exposition (DANCE)*, (San Francisco, CA, USA), May 2002.

[23] "Squid Web Cache Proxy." URL http://www.squid-cache.org/.