Efficient and robust Streaming Provisioning in VPNs

Z. Morley Mao[*], David Johnson, Oliver Spatscheck, Jacobus E. van der Merwe, and Jia Wang
AT&T Labs–Research

ABSTRACT

Today most larger companies maintain virtual private networks (VPNs) to connect their remote locations into a single secure network. VPNs can be quite large covering more than 1000 locations and in most cases use standard Internet protocols and services. Such VPNs are implemented using a diverse set of technologies such as Frame Relay, MPLS or IPSEC to achieve the goal of privacy and performance isolation from the public Internet.

One of the services which recently received increased interest in the VPN space is the use of VPNs to distribute proprietary live streaming content. For example, a VPN could be used to broadcast a CEO employee town hall meeting. To distribute this type of content economically without overloading the network, the deployment of streaming caches or splitters is most likely required.

In this paper we address the problem of optimally placing such streaming splitters or caches to broadcast to a given set of VPN endpoints under the constrains typically found within a VPN. In particular, we introduce an algorithm which guarantees the optimal cache placement if interception is used for redirection. We prove that the general problem is NP hard and then we introduce multiple heuristics for optimal cache placement in the general case which we then evaluate using extensive simulations.

1. INTRODUCTION

The distribution of live streaming content is gaining popularity in VPN environments to support Webcasts ranging from tens to hundreds-of-thousands of streaming media clients. Due to a lack of layer 3 multicast availability in these environments, these streaming services are typically offered using unicast transport protocols. Obviously the inherent problem of distributing such a large number of streams via unicast delivery is that some links within the VPN might become overloaded.

The typical VPN network topology compounds this problem even further. VPNs usually consist of a a small hub network which provides connectivity to in some cases thousands of individual spokes. The fact that both the hubs in the hub network and the spokes are usually geographically diverse makes increasing the bandwidth to accommodate a large number of unicast streams prohibitively expensive.

The standard solution to this problem is to deploy streaming cache servers within the VPN which in the context of live streaming simply split the unicast traffic and, therefore, can be used to offload congested links. Such cache servers are offered by a large number of vendors such as Network Appliance, Volera or Cisco Systems. However, the cost of the caches themselves as well as the maintenance cost associated with each cache requires a careful placement of the caches to minimize the overall cost of the live streaming infrastructure. In the context of an existing VPN it is therefore important to find the optimal cache placement as to minimize the number of caches which have to be deployed to serve a given client population.

We address this problem by first proving that the optimal solution to the problem is NP hard. We then show that the problem restricted to the use of interception caches has a provable optimal solution with complexity $O(Elog(V))$ with E being the number of links and V being the number of routers in the VPN. We also provide heuristic solutions for the general case and thoroughly evaluate our solutions using simulations.

The rest of this paper is organized as follows. In Section 2 we discuss related work. Section 3 contains a more detailed problem statement and describes our proposed cache placement algorithms. Section 4 describes the simulation method we used to evaluate the proposed algorithms and Section 5 contains the simulation results. Section 6 concludes our paper.

2. RELATED WORK

While a significant body of related work exists, we believe the work presented in this paper to be unique in the following aspects:

- Considering the placement of cache servers for the support of live streaming media in a VPN environment. Most related work deals with the placement of servers to support Web traffic in an Internet environment.
- Considering the minimum number of caches to fully support a known user population. Most related work considers the potential performance improvement of users as a tradeoff against the cost of deploying servers.
- Considering the robustness of our algorithms in the face of imperfect input data. While some papers mention this as an issue, we are unaware of any work in which is has been thoroughly examined.

In the Web domain several papers consider the placement of objects or replicas on network servers [7, 9, 8]. The purpose here is to decide which objects should be replicated on what set of network servers while trading off improved performance against the cost of servers. For the problem of live streaming which we have addressed this kind of placement is not possible because in general all of the

[*]Zhuoqing Morley Mao (email: zmao@cs.berkeley.edu) is a Computer Science graduate student at University of California, Berkeley. This work was done during her internship at AT&T Research Labs.

content (i.e. the complete live stream) needs to be distributed to all users.

Another group of papers investigate the placement of network servers in various scenarios [12, 11, 6]. The most general of these with a problem statement closest to our work is [12]. They consider the number of servers which are needed and where they need to be placed in an overlay network to satisfy a certain quality of service to all clients. Their work differs from ours in that it aims at an Internet environment rather than the more constrained VPN environment we have considered. Furthermore, while their problem statement is very general, they make a number of simplifying assumptions, e.g. capacity constraints are ignored, which is not realistic for our problem setting. Both [6] and [11] investigate server placement in the context of performance improvement of Web traffic.

3. PROPOSED SOLUTIONS

3.1 Problem statement

Figure 1 depicts a typical VPN topology to illustrate the scenario we are considering in this paper. In the figure a live stream is assuming to originate from an origin server which connects directly to a "hub-network". The hub network consist of lower capacity (typically WAN) links and is densely interconnected. The hub network interconnects "spoke networks" consisting of more richly connected islands of typically higher capacity.

Figure 1 also shows the inherent problem with distributing continuous streaming content via unicast delivery. The thick lines show a shortest-path distribution tree from the origin server. The numbers next to each of the lines indicate the required capacity of each link assuming that each stub network requires three units of capacity. The well known solution to this scalability problem consist of the distribution of streaming caching servers throughout the network to effectively form an application level multicast distribution mechanism.

In this paper we address the practical and important problem of determining the minimum number of such caching servers required to deliver live unicast streaming content to a given client population. We start off by assuming that the network topology and link capacities are known. These constraints are relaxed later to investigate the robustness of our algorithms. We assume all clients need to be satisfied, i.e. we want to serve the live stream(s) without degradation in quality to all clients. Given the minimum number of caching servers, a second optimization criterion is minimizing the total bandwidth usage by placing the caches strategically.

There is an inherent tradeoff between bandwidth utilization and the number of caches needed to satisfy a user population. At an extreme, we can place a cache at each router to achieve the minimal total bandwidth usage, i.e. emulating the use of native multicast. The optimization criteria we study in this paper requires that we satisfy all users with the minimal number of caches. Therefore, we first find the minimal number of caches and then place them to minimize bandwidth usage. We emphasize that our algorithms can be easily adapted using a different cost model by modifying the cost of adding cache and bandwidth usage. For instance, our algorithms can be used in discovering where the bottleneck links are and compare the cost of placing a cache versus just upgrading the link. Our contribution lies in providing a general framework for optimizing different criteria. We also believe that our chosen criteria represents a reasonable choice for practical considerations.

More precisely we make the following assumptions unless otherwise stated:

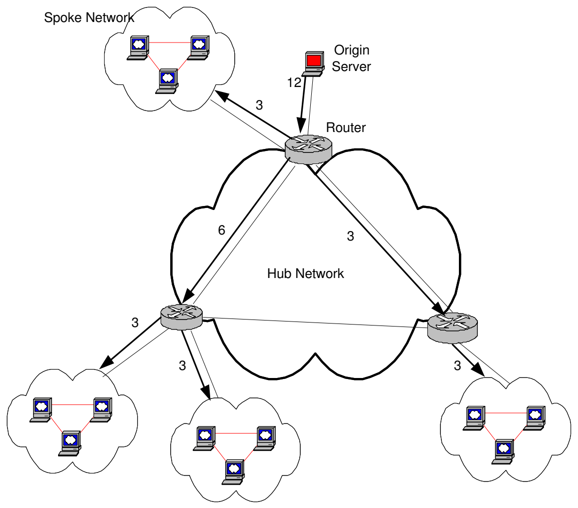

Figure 1: Streaming distribution in a VPN from the origin server

- Network information: The topology, link capacity and receiving client distribution is known.
- Origin streaming server: The location of the origin server and the bandwidth of the stream are known.
- Routing: IP layer routing is performed using a shortest path algorithm. We assume the default routing metric (link weight is the reciprocal of the link capacity) and assume the absence layer 3 multicast support.
- Request routing: A client can request the stream from any cache or the origin server.
- Cache location: any router can be a potential location for cache placement.
- Application requirement: The delay between the client and the cache/origin server is not of concern, however, the route must provide sufficient bandwidth for the stream.
- Bandwidth usage: The bandwidth usage for each link cannot exceed the link capacity. We assume bandwidth on different links are of equal cost and we minimize the total bandwidth used. If different types of links have different cost models, the problem can be easily modified by including a cost unit for each link.
- Cache or origin server is not the bottleneck relative to the bandwidth usage, thus, server load is not a consideration in our algorithm. We assume that each server can saturate the total bandwidth of its outgoing links. This assumption is reasonable in the streaming context, as small streaming servers today can saturate OC-12 links trivially and large servers can handle Gigabit speeds. Note, however, our solution can be easily modified if server load is an issue by including a bound on the server load.
- VPN topology is of hub and spoke nature, where the hub consists of a few highly meshed routers connected with very high bandwidth links across the wide area. Each hub node is serving several stub domains.

As indicated earlier, we use this "perfect" knowledge as a starting point and then consider the robustness of our algorithms in the case of incomplete knowledge.

3.2 Problem complexity

The problem of finding a minimum cache placement to satisfy a given user demand is an NP hard problem. Here we formally show how a well known NP hard problem can be reduced to our problem.

Minimum cache placement problem: Given a network modeled as a directed graph $G(V, E)$ with nodes V, and directed links E of known capacity. Each node has a demand. The root $R \in V$ of the distribution is given and each node must have its demand satisfied. Find the minimum number of caches and their placement to satisfy all demand.

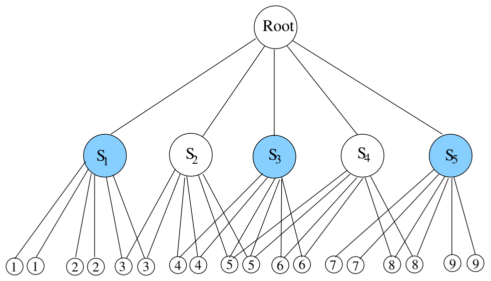

Figure 2: Reduction from exact cover by 3-sets (X3C) with additional constraint that no element occurs in more than three subsets to minimum cache placement problem

We reduce the exact cover by 3-Sets(X3C) [5] to our problem. An instance of X3C is a set X ($|X| = 3q$) and a collection C ($|C| = n$) of 3-element subsets or triplets of X. There exists a solution for X3C if and only if there exists a subcollection $C' \subseteq C$ such that every element of X occurs in exactly one member of C'. It's clear that such subset C' must contain exactly $|X|/3 = q$ triplets. The problem remains NP-complete with the additional constraint that no element occurs in more than three subsets.

We reduce the X3C instance with this constraint to an instance of minimum cache placement problem as follows. We first construct the graph by adding a vertex for the origin server and for each of the n sets and *two* vertices for each of the three elements of the set as shown in Figure 2. We add an edge between the origin server and all the vertices representing the sets, an edge between each set node and all nodes representing elements within the set. Each edge represent two bidirectional links with capacity 1. We assume that each element vertex has a demand of 1, i.e., a user, that must be satisfied either from a cache server or the origin server. There are total $6q$ element nodes representing $6q$ users.

We select q nodes from the n vertices representing the sets to be the set cover. We now prove that all $6q$ users can be satisfied if and only if the 3-set collections C has an exact set cover C' representing the q servers. We first show that if all users are satisfied, the nodes representing q servers must be a solution to X3C. Each set node has degree 7: one link to the root and six links to the element nodes. Since the root is by definition a server, each set node can only serve at most 6 users. There are q servers and $6q$ total users, each user therefore is only served by exactly 1 server. This means that q servers represent a solution to X3C.

We now show if there exists a solution to X3C, the servers representing this solution are sufficient to satisfy all $6q$ users. If we place all the servers at the set nodes, this can be guaranteed because of the definition of X3C. We now show that if we place any server at the element node, we cannot satisfy all users. Consequently, our assertion must be true. A server at a server node can serve all 6 users represented by elements in its set. However, a server at the element node can serve at most 4 users, because it has at most degree 4: 1 link to its own user and three links to set nodes representing the sets it belongs to. To satisfy $6q$ users using only q servers, each server must serve exactly 6 users. Therefore, no server can be placed on the element nodes. Figure 2 shows an example where $\{S_1, S_3, S_5\}$ represents both a solution to X3C and also satisfies all $6q = 18$ users.

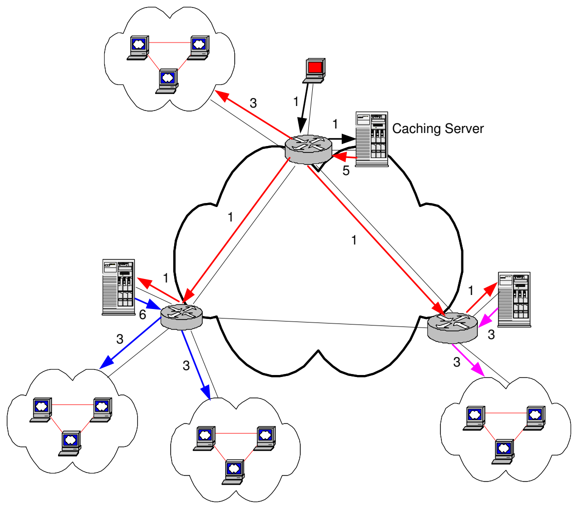

Figure 3: Streaming distribution using interception proxies

We have shown that an instance of X3C can be reduced to our problem; therefore, our problem is NP-hard. Notice that shortest path routing is used in the construction of our proof; therefore, the problem remains NP-hard under the shortest path routing constraints, e.g., routing using OSPF.

3.3 Interception proxy based solution

In order to make use of streaming caching servers to build application level distribution trees, streaming clients have to request the stream from a caching server rather than the origin server. One way in which this can be achieved is by transparently intercepting requests for streaming content at the network layer and passing it to a caching server, the intercepting proxy approach. WCCP [3] is an example protocol that enables this functionality by intercepting requests on certain port ranges in a router and passing all such traffic to a caching server.

As shown above, the problem of finding the minimum number of caches to satisfy a given user demand is NP-hard. Intuitively, the complexity arises from the fact that whenever a new cache is placed, a new distribution tree can be built potentially utilizing links that may not have been available before and this affects the selection of server for each user to minimize bandwidth usage. Thus a placement of a cache can completely change the traffic flow pattern and the final flow pattern may not be a tree.

Fortunately, when intercepting proxies are used, the problem is simplified by restricting the traffic flow on a single distribution tree built from the origin server. Thus, a cache can only serve the clients downstream from the tree, given the view point that the root of the tree is the origin server. Such a restriction represent the mechanism used in the intercepting proxy solution where end clients are requesting from the origin server as they would normally do, but caches placed on the route between the client and server intercepts the request and delivers the stream. We now present a simple and efficient greedy algorithm that gives the *optimal* solution in terms of the minimum number of caches. We subsequently show how a simple dynamic programming algorithm working on the results of the greedy solution gives the *minimum* bandwidth usage for the minimum number of caches.

3.3.1 Greedy minimum cache placement algorithm

The topology is an undirected graph, $G(V, E)$, each edge $e \in E$ labeled with link capacity (C). Each vertex v denotes a router which has n_v stream requesting clients attached to the router. v_s is the origin server.

Construct the shortest path tree from v_s using dijkstra – a directed graph, $G'(V, E)$. Assuming all clients initially request from the origin server, label each link e with a demand for the bandwidth D. Identify links e' whose demand D exceeds capacity C, i.e, $D > C$, let v_i' be the destination router of the flow on the saturated link e_i'. let $\{V'\}$ to be the set of destination routers for all saturated links. For ease of explanation, we call these nodes *red* indicating overloading, the rest of nodes are *green*.

Note that $\{V'\}$, the red nodes are our candidate locations for placing cache servers. If we place a cache at each of these router, then we can satisfy all clients; however, this may not be the minimum number of caches.

Label each node v_d with a depth value d – the number of hops away from the origin server. Let d_{max} be the maximum depth from the origin server.

//starting with red nodes farthest away from source
for each $i = (d_{max}, d_{max-1}, ...1)$,
let $\{V_i'\}$ be the set of routers, i hops away from the origin server, that belong to $\{V'\}$, i.e., the red nodes or candidate locations for cache placement at depth i. Place a cache at all red nodes at depth i, change the demand on the links upstream.

This algorithm walks up the tree from the lowest depth and adds the caches at decreasing depth when needed. It only adds a cache when it is absolutely required at a given depth. The tree is traversed from the largest depth from the origin server, because placing a cache serve at a larger depth reduces the bandwidth utilization at the links at smaller depth and may eliminate the need of caches at small depth. The running time of the algorithm is $O(Elog(V))$, given that the depth of a balanced tree is $O(log(V))$ and at each depth, we visit each edge at most once. For unbalanced tree, the depth can be V; so the worst case running time is $O(V * E)$, still very efficient for any large topologies. The preprocessing of building distribution tree using dijkstra's algorithm and labeling the demand on the links are negligible compared to our algorithm.

We now prove that the simple greedy algorithm gives the optimal solution – the minimum number of caches for the network for known user demand assuming the flow is restricted to a single distribution tree from the origin server. We prove by induction and show that the problem has two ingredients [4] exhibited by most problems solvable using a greedy strategy: the greedy-choice property and optimal substructure.

Proof: The greedy choice is picking the red nodes at the largest depth – $x \in \{n_{d_{max}}\}$ to be cache locations, given the intuition that they will reduce the most amount of total demand on all links of the distribution tree. In our algorithm, we do not care which particular order caches are placed at the red nodes at a given depth. We only care that the red nodes are picked in decreasing depth. For efficiency reasons, it makes sense to place caches at all red nodes at a given depth to reduce the demand of upstream links.

We first show that there exists an optimal solution that begins with a greedy choice, i.e, with x. Assuming there exists an optimal cache placement sequence A that starts with n_k. If $k = x$, then the optimal solution begins with a greedy choice. If $k \neq x$, we want to show that there is another optimal placement sequence B that begins with the greedy choice x. Let $B = A - \{k\} \cup \{x\}$. Placing a cache at the largest depth always reduces the upstream bandwidth and never cause more caches to be placed anywhere – including upper level. Furthermore, placing k first does not eliminate the need to place x, because k does not alleviate the load on links located at larger depth than k's depth. Therefore, there always exists an optimal minimum number of cache placement that starts with red nodes at the largest depth: x.

Moreover, once the greedy choice of picking red nodes at the largest depth is made, the problem is reduced to finding the minimum number of caches compatible with the first choice. Note, adding a cache at the largest depth always reduces the bandwidth consumption. We show by induction that if A is an optimal solution to the tree S, then $A' = A - \{x\}$ is optimal to the subtree S', which is the original tree S deleting the nodes at largest depth and links connecting between them and other nodes. We prove by contradiction this is true, assume we could find B' to S' with fewer caches than A', then adding $\{x\}$ to B' would be a better solution to S, contradicting the optimality of A.

3.3.2 Minimum bandwidth usage

Given the cache placement obtained from the greedy algorithm which gives the minimum number of servers, we can use a dynamic programming algorithm to place them in such a way that minimizes the bandwidth usage. The placement obtained using the greedy algorithm has the property that caches cannot be moved up the tree, if we view the top of the tree as the root of the distribution. However, moving caches down the tree may create bandwidth saving, because the parent node of the cache needs only to deliver 1 unit of the flow to the cache. For a given cache, we decide whether and where to move it down based on the amount of bandwidth saving obtained. The bandwidth saving is calculated by considering the amount of increase in bandwidth due to sibling nodes of the candidate cache location and bandwidth decrease due to the demand of the cache location itself. We also need to keep track of the minimum available bandwidth between the current node and its next server up in the tree to make sure that the bandwidth increase does not cause any link capacity to be exceeded. Since caches can only be moved down the tree, any subtree can only have increasing number of caches. The formulation gives rise to many overlapping subproblems and dynamic programming can be used to give an optimal solution. The running time of the algorithm is $O(VlogV)$, since there are at most V caches and each can be moved at most $logV$ depth steps down the tree if the tree is balanced. For unbalanced tree the running time is $O(V^2)$.

In our simulation study (Section 5), clients are uniformly distributed across all routers and the distribution tree generated are fairly balanced. Therefore, we found at most 10% improved in bandwidth saving using the dynamic programming algorithm. Intuitively if there exists more imbalance in the distribution tree, the algorithm will move caches towards subtrees where there is higher demand and generating more bandwidth savings. Thus, our greedy algorithm is quite good in terms of bandwidth usage for topologies with relatively uniform client distribution and link connectivity.

In summary, by using interception based redirection which restricts the traffic flow along a single distribution tree, efficient optimal algorithms exist in finding the minimum number of caches and their placement to achieve minimum total bandwidth usage. Such a

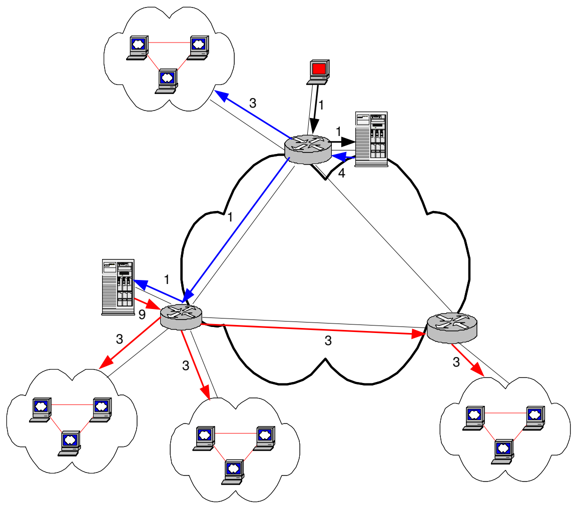

Figure 4: Streaming distribution with router-based redirection

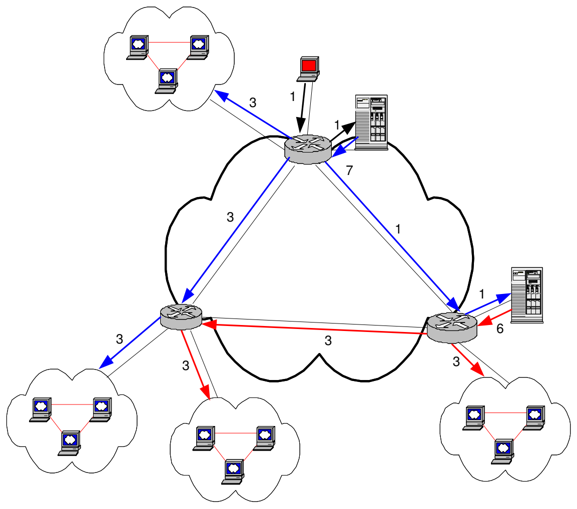

Figure 5: Streaming distribution with client-based redirection

solution requires minimum change in the infrastructure and can be easily implemented by placing caches to intercept the client stream requests and requires no modification at the client.

3.4 Additional redirection systems

In the previous section we assumed that client requests are intercepted on layer 3 and redirected to an appropriate cache. However, all cache vendors offer in addition to interception based redirection layer 7 based redirection methods such as DNS, HTTP/RTSP redirect or meta-file rewriting [2].

Therefore, layer 7 redirection relaxes the constraint of using a single distribution tree from the origin server and allows all clients to use any cache, no longer restricting the usage to the cache to only downstream clients from the cache in the original distribution tree. Since we proofed this problem to be NP-hard as show in Section 3.2, we can only use heuristics to attempt to find solutions close to the optimal.

Layer 7 redirection can be done at different granularities with higher cost associated with a finer granularity. At the extreme, each client attached to the router can potentially select a different cache server. A more coarse-grained redirection can be done on a per prefix level. Similarly, clients attached to the same router are constrained to select the same server. We consider the improvement in cache reduction compared to the greedy algorithm at two different granularities: client-based redirection and router-based redirection below. Note, due to the inherent tradeoff between bandwidth usage and the number of caches, these solutions attempt to find fewer caches by making use of links with spare bandwidth and hence have higher overall bandwidth utilization. These two approaches are shown in Figures 4 and 5. In Figure 4 all clients connecting to the same router are assumed to be redirected to the same caching server, while in Figure 5 redirection happens at a finer level of granularity along some clients to be served from a different caching server.

3.4.1 *Router-based redirection*

The framework of original greedy algorithm (Section *3.3.1*) is retained and a simple modification is made to maximize cache utilization. We still walk the original distribution tree rooted at the origin server bottom up, from the leaves to the root. Instead of placing a cache at each red node at a given depth, we place a cache one at a time by selecting the red node to be the cache that maximizes the number of other red nodes that can turn green because of requesting from the cache. We sketch the algorithm below.

For each red node at depth i – v_i belong to all red nodes $\{V_i'\}$, define x_i to be the *estimated* maximum number of other red vertices in $\{V_i'\}$ that can have enough of its children (clients or routers) request from v_i to become green. This is done by examining if the extra demand (D-C) for the red node can be served by v_i, (we need to consider the new routes between the children nodes to v_i).

The estimated maximum x_i is calculated by picking randomly the vertices in $\{V'\}$ in decreasing depth, starting with the same depth. Every time, if a red node picked can have enough of its clients or children nodes request from v_i and become a green node, then the affected links' demand is recomputed to reflect the change. Note, this is an estimated max – the actual max number may be larger, because the order of picking the vertices may matter.

Now, pick v_i with largest x_i, i.e., is estimated to change the maximum number of red nodes to green. Update the demand, and update the x_i for each $\{V_i'\}$, repeat again, until $\{V_i'\}$ is empty, i.e., all nodes are green at depth i. Then, we go to the next smaller depth i.

The running time of this algorithm is $O(V^2E)$, because in the worse case we need to consider all other nodes, when deciding in placing a cache at a given red node. The basic idea behind this algorithm is the greedy strategy in placing a cache at a red node in decreasing depth that can help alleviate maximum number of other red nodes by allowing other red nodes to request from the newly placed cache. Note however, the solution may give the same number of caches as the greedy strategy but with increased bandwidth utilization due to the strategy in spreading the load on other distribution trees rooted at caches.

3.4.2 *Client-based redirection*

In the above algorithm, we assume that all clients attached to the router must request from a single cache server. A simple modification can be made such that we try to satisfy the maximum number of clients of a red node. Thus, there is no need for a cache server to satisfy all clients for a given node. This way, the unit of flow is a single user rather than a collection of users. As a result, bandwidth can be more efficiently utilized.

3.4.3 *Local exhaustive search*

To evaluate the effectiveness of the above two heuristics, we would like to achieve an lower bound on the minimum number of caches needed for a given network assuming any user can request from any cache. Due to the NP-hard nature of the problem, exhaustive search on the entire topology of any reasonable size would take exponential amount of time and is infeasible in practice. Instead, we obtain an approximate bound by performing in parallel a local exhaustive search on each stub domain, and finally on the hub domain given the bandwidth requirements from the stub domains. The motivation is that wide area links between the hub node and the stub domain are usually more expensive than local area bandwidth within the stub domain. Therefore, it makes sense to place caches within a stub domain to satisfy all its user demand. Consequently, we restrict the users of the cache to be those within the same domain.

To minimize the amount of the time of exhaustive search on each local domain, we use the heuristic of only considering stub domains where there are at least two caches needed based on the result from the greedy algorithm. Furthermore, we try to first place caches at nodes with highest outgoing bandwidth. As we show in Section 5, both our client-based and router-based redirection heuristics most of the time perform better or close to this algorithm. This demonstrates that our simple heuristics provide quite good lower bound on the number of caches needed.

3.5 Robustness considerations

One of the assumption in our problem statement is that we have complete knowledge of the network including the topology, link capacity, sharing of layer 3 links at layer 2. However, in practice, it is difficult to obtain completely accurate information on the entire VPNs. Wide area links are typically well-known, but various locally managed stub networks may have network configurations that are not fully known. Link bandwidth estimation tools may have errors and do not provide information on layer 2 sharing due to VLAN trunking. Furthermore, background traffic load can always reduce the available bandwidth.

Thus, it is essential to evaluate how well our algorithms perform under such practical constraints. The solutions presented sofar unrealistically assume that the link capacity is well known and there is no background traffic. We slightly modify our algorithms by avoid overloading the links by more than 80%, assuming the error bound of available bandwidth estimation is within 20%. These results are also shown in Section 5.

4. SIMULATION METHODOLOGY

To evaluate the effectiveness of various algorithms proposed in this paper, we simulate the behavior of our algorithms on typical VPN hub-and-spoke topologies. In this section, we discuss the simulation parameters and assumptions, network topologies, and evaluation metrics used.

4.1 Simulator design

We implemented our algorithms in C using the Stanford Graphbase library [10] which provides the basic graph data structure and graph manipulation routines. Our simulator is very efficient and can simulate up to 10,000 routers given the 2 Gigabyte memory constraint on the machine we use for simulation. The design of the simulator is modular, allowing the specification of different cost constraints for different optimization criteria in the algorithms.

4.2 Network topology

Table 1: Link capacity assignment in simulations

Link type	10Mbit	100Mbit	1Gbit	T1	DS3	OC12	OC48
spoke	40%	50%	10%	-	-	-	-
hub-spoke	-	-	-	45%	30%	25%	-
hub	-	-	-	-	50%	45%	5%

Table 2: Error distribution for client-based capacity estimates (unit:Mbps)

sample size	mean	median	variance	max	min	86%tile	95%tile
619	-21.5	-0.028	0.0071	291	-971	0	22.2

VPNs typically have hub and spoke topology, where the hub consists of a group of highly meshed routers spread across the wide area with high link bandwidth between them. From each hub router, there are several spoke domains hanging off it. Each spoke domain represents a single site and typical consists of up to a dozen routers. For redundancy purposes, a spoke domain can also be multihomed to another hub router at a different location. The wide area link between the spoke domain and the hub router are also typically quite high, but usually of lower speed than the links between hub routers. Finally the links between routers in the same spoke domain are typically 10Mbit, 100Mbit or 1Gigbit Ethernet links. Our simulations encompass three classes of VPNs.

- Large companies: large number of spoke sites with high bandwidth between the hub routers.
- Retail stores: large number of spoke sites with relatively smaller bandwidth between hub routers.
- Engineering firms: a few dozen site with hundreds of networks at each site.

We modified the GT-ITM internetwork topology generator [1] to produce layer 3 hub-spoke topologies. Since each router has a number of clients requesting streams, we make sure that the topology is connected. The topology inside each spoke domain are constructed by connecting two vertices randomly with a specified probability.

The input parameters to the topology generator are the following: number of fully meshed hub routers, average number of spoke domains per hub router, average number of routers per hub, number of multihoming links, and the probability that two routers inside the same spoke domain is connected. Multihoming links are randomly placed between a hub router and a spoke router not within the spoke domain connected to the hub router. Furthermore, for evaluating of the resilience of our algorithms to uncertainties in topology data, the topology generator also takes in a error distribution for link capacity estimation and a value representing the probability that two links in the same spoke domain share at layer 2.

4.3 Simulation parameters and assumptions

We now describe the basic assumptions made in the implementation of our simulator. Some of these assumptions are specified as input parameters and can be modified for different optimization goals.

- Link capacity and routing: The network parameters are mostly known, both link capacity as well as Layer 2 sharing of links. Symmetric shortest path routing is assumed with link weights being the reciprocal of the capacity values. The links are modeled as bidirectional with symmetric bandwidths. We relax the assumption of known capacity and link sharing in the evaluation of robustness of our algorithms. We use the distribution in Table 1 to assign link capacities. The values in the table present very typical values for these types of links.

- Client distribution: clients that need to receive the unicast multimedia streams are located at known locations. Each router has between 20 and 70 clients attached to it, which is typical distribution for any LAN. Each client must receive the stream either from a stream source – an origin server or a cache server. A single stream requires 300kbps bandwidth in the simulations.

- The streaming source is located in the hub domain. If a different streaming source is used, the solution of the problem is still applicable by first delivering the stream to the intended origin server location and then use the allocated cache servers to distribute the content as is often done in practice. From simulation studies, we found that due to the symmetric nature of the hub domain topology where hub routers are highly meshed, the cache placement results are quite stable for different selection of streaming sources within the hub domain.

- The error distribution for link capacity estimates is directly obtained from a measurement study we performed on the AT&T corporate VPN. We developed both Java and activeX based client side measurement tools and asked voluntary participants from AT&T corporate employees to visit the measurement Web page. We compare our measurement results with the survey results from the participants to obtain the error distribution shown in Table 2. Most of our estimation underestimates the true capacity values; therefore, the errors do not have much impact on overloading links.

4.4 Evaluation metric

We evaluate our algorithms in terms of the number of caches required to satisfy a given user demand. Secondly, we examine the amount of bandwidth used. Due to the inherent tradeoff between number of caches and bandwidth usage, the fewer caches usually result in more bandwidth usage. However, we found that some algorithms result in the same number of caches but with higher bandwidth usage. To test the resilience of the algorithms to inaccurate topology information, we calculate how many users are not satisfied.

5. SIMULATION RESULTS

In this section, we evaluate the effectiveness of our proposed algorithms on a variety of hub and spoke topologies. We provide insight on when more complex solutions will provide more benefit than simple greedy strategies. We also show how various properties of the topology affect the results of the algorithms.

5.1 Interception proxy based solution

5.1.1 Greedy minimum cache placement algorithm

We first study the effectiveness of our greedy algorithm. We also highlight some interesting effects of the topology on the cache requirement. Figure 6 plots the number of cache servers required with two spoke router degree values and increasing spoke domain size given a fixed number of hub size and number of spoke domains per hub router. We define the *spoke router degree* or *spoke degree* to be the average number of outgoing links a router in the spoke domain has. The figure shows that increasing spoke domain size increases cache requirement and increasing spoke router degree decreases cache requirement. With larger spoke domain size, inherently, there are higher demand for bandwidth because higher number of clients. With higher router degree, a cache placed inside spoke domain can serve more clients; therefore, fewer caches are needed overall. Intuitively, caches should be placed where there is high degree of connectivity to maximize its benefit.

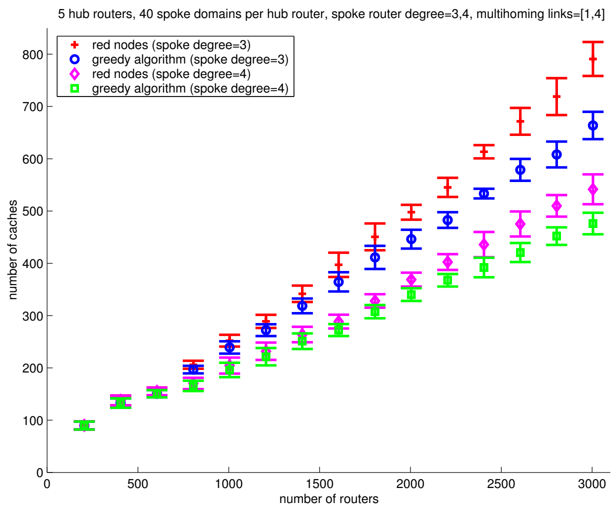

Figure 6: Effect of spoke domain size and spoke router degree on cache requirement

Figure 6 also plots the number red nodes – routers with demand from child routers exceeding its parent link capacity in the tree. This gives an upper bound on the number of caches required using the greedy algorithm. As the number of routers increases, the probability of a cache alleviating the demand of its parent red nodes also increases. Therefore the difference between the number of red nodes and caches increases with router count. The variance in the results are due to differences in the number of multihoming links, host distribution, and randomness in topology generation.

Figure 7 plots the total bandwidth requirement in the presence of caches using the greedy placement (Figure 6) comparing with that in the original network using a distribution from the origin server. We define *total bandwidth* to be the sum of bandwidth on all links. The benefit of cache servers is quite clear, as the bandwidth requirement grows roughly linearly with the greedy algorithm's cache placement. We also note that the original bandwidth requirement for the topology with spoke router degree of 3 is slightly higher than the topology with spoke router degree of 4 for the same router count. Intuitively, with higher router degree, there is higher chance of aggregating traffic because the distribution tree is bushier. With higher router degree, a single cache also has more chance of serving more clients. As we have shown in Figure 6, few cache servers are needed. Due to the inherent tradeoff between number of cache servers and bandwidth usage, the bandwidth usage in presence of caches is slightly higher for the topology with spoke router degree of 4, because of fewer caches.

Figure 8 shows the median of the number of caches required for a given topology as the number of multihoming links increases. On average there is less requirement for cache servers as multihoming increases. Any multihomed spoke domain can reach the hub network through either of the two links. The tree constructed by

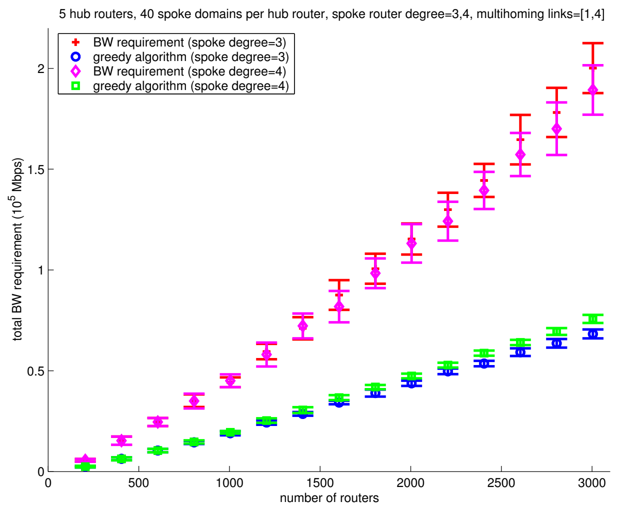

Figure 7: Effect of spoke domain size and spoke router degree on bandwidth requirement

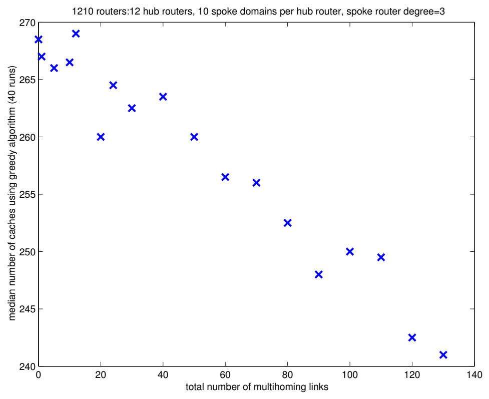

Figure 8: Effect of multihoming links on cache requirement

the greedy algorithm rooted at the origin server in the hub network takes advantage of such multihomed links to minimize the number of caches needed. In this example shown, for the same topology size, the amount of saving in the number of cache server requirement can be more than 10%.

Figure 9 shows the effect of the number of spokes per hub node or the spoke domain size have on cache requirement. In this set of simulations we keep the total number of routers in the topology constant at 2412; the total number of hosts are between 105486 and 109072 with standard deviation of 694. There are 12 hub nodes in the hub network. We vary the number of spoke domains attached to each hub router. In effect, we also vary the number of routers in each spoke domain to keep the total number of routers constant. At one extreme, each hub router has only a single spoke domain with 200 routers. At the other extreme, each hub router has 200 spoke domains, each with a single router. As the number of spoke domains increases, there is less aggregation possible. Each spoke domain only has a single link to the hub domain. If that link is unable to support the client demand within the spoke domain, a cache must be placed in there. The smaller the spoke domain, the fewer clients can make use of the cache. As a result, the topology with

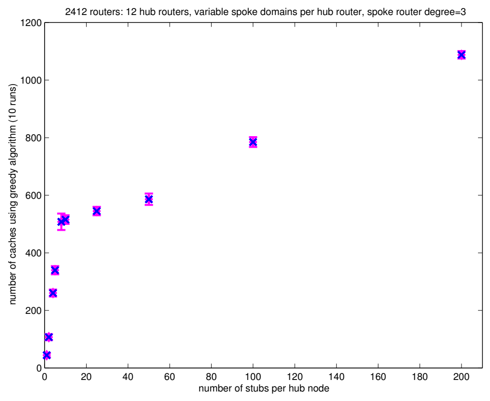

Figure 9: Effect of number of spoke networks per hub router or spoke size on cache requirement

numerous small spoke domains require more caches. We emphasize that this is an inherent limitation in the topology and is not due to the greedy algorithm. Designers of VPNs may take this effect of topology on cache requirement into consideration.

5.1.2 *Minimum bandwidth usage*

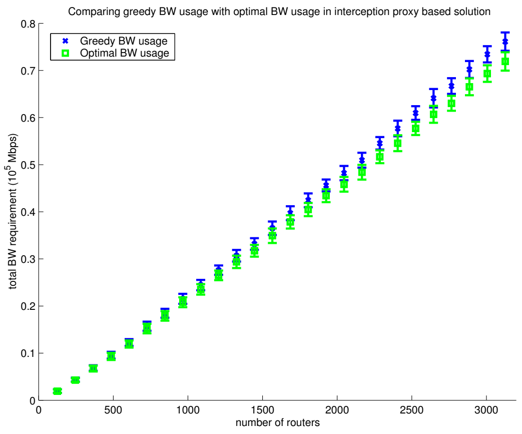

Figure 10: Comparing greedy's bandwidth usage with the optimal bandwidth usage in interception proxy based solution

Figure 10 shows the optimal bandwidth usage using interception proxy based solution from the dynamic programming algorithm described in Section 3.3.2 compared with that of the greedy algorithm. The topology consists of a 6 router hub network, with an average of 20 spoke domains per spoke router, each spoke router has an average degree of 3. The result for other topologies are very similar. This shows that the greedy algorithm is already quite good in optimizing bandwidth usage. The maximum improvement for this set of simulations is 20%. On average, the improvement is only 4.1%.

5.2 Additional redirection systems

We now compare the greedy algorithm with the additional redirection systems. Figure 11 shows that as the number of routers increases greedy is outperformed by both router-based and client-based redirection. With large topologies, Client-based redirection

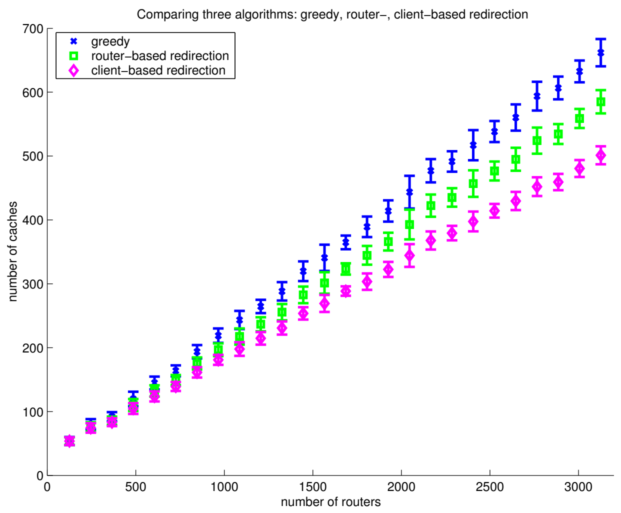

Figure 11: Comparing greedy algorithm with router and client based redirection

can reduce the number of caches used by greedy algorithm by more than 27%. We also observe that finer grained redirection provides sufficient benefit for large networks. With client-based redirection, the number of caches can be reduced by more than 17% comparing with router-based redirection. This particular example shows the network topology consisting of six fully meshed hub routers in the hub network, each hub router with on average 20 spoke domains, and average spoke router degree of 3. The number of multihoming links vary between 1 and the hub size 6. We have observed similar results with a different topology setup.

5.3 Robustness evaluation

To evaluate the robustness of these algorithms, we introduce some errors in the topology information, specifically, the link capacity estimation and sharing of links at layer 2. We obtain the error distribution of capacity estimation from a measurement study we conducted shown in Table 2. The measurement methodology we used is a variant of the standard packet pair based probing. We note that other types of measurement methods may have different error characteristics. In the future, we plan to consider other types of error distribution. We set the probability of sharing between two random links inside the same spoke domain at layer 2 to be 0.1 for the purpose of this study.

5.3.1 Existing algorithms

Figure 12 plots the number of unsatisfied users due to overloaded links for each algorithm. The number of such users increases with the topology size as the number of highly loaded links increases. However, the overall percentage of unsatisfied users never exceeds 6% in all three algorithms. The particular topology under this study consists of a hub network of 5 routers, each with an average of 20 spoke domains and spoke degree of 3, without any multihoming links. Greedy seems to be most error resilient due to the fact that it places more cache servers in the network and thus has fewer loaded links. Inherently, all three algorithms are vulnerable to inaccurate input data, because each tries to find links with spare capacity to receive video streams. For each algorithm, the maximum link utilization is usually close to 99%.

5.3.2 Robustness improvement

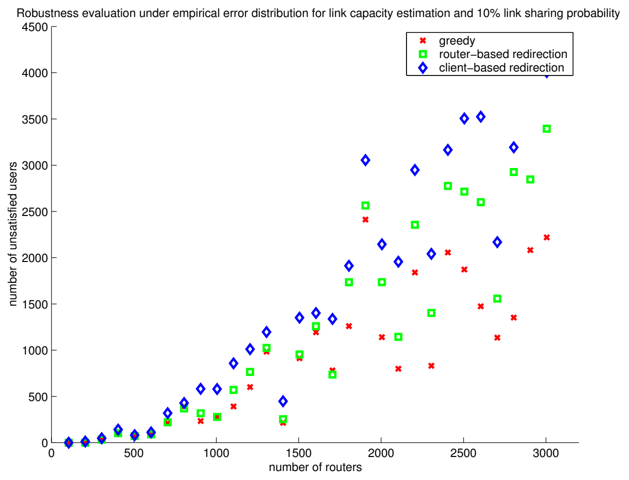

Figure 12: Understanding the error resilience of the algorithms to imperfect topology data

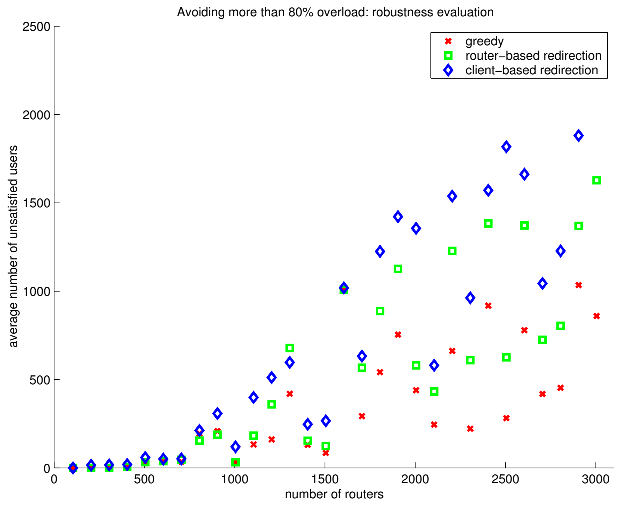

Figure 13: Evaluation of robustness heuristics to imperfect topology data

One possible way to deal with the imperfect topology information is to always leave some spare bandwidth on each link or on links where we suspect our capacity estimation is incorrect. We implemented in the simulator this heuristic and set the amount of spare bandwidth to be 20% of the estimated link capacity. We observe in Figure 13 that the overall trend of increasing number of unsatisfied users with increasing topology size remains. However, based on the y-axis, the number of unsatisfied users is reduced. Our evaluation also makes it compelling that we need measurement information or external knowledge on where links are most likely shared and where capacity estimates are not accurate. Given such information, we can selectively avoid overloading these links in the algorithms. Note, the penalty of underestimating link capacity results in requiring more caches than actually needed.

6. CONCLUSION

In this paper, we study the problem of placing cache servers in VPNs to provision for unicast based video streaming events. Our goal is to satisfy a given client population with the minimal number of cache servers. Given the bound on the number of cache servers, we add the additional goal of placing them in such a way as to minimize the total bandwidth usage. We developed provably optimal algorithms to achieve both goals using an interception cache server based solution. In addition we prove that the problem is NP-hard in general. We then develop a set of simple and efficient heuristics to provide reasonable solutions to the cache placement problem if non-interception based redirection is used.

In addition to those theoretical results we performed extensive simulation to evaluate the performance of our algorithms in a realistic setting. We discovered in particular that if non-interception based redirection systems are used the number of caches can be reduced by more than 27% using our heuristics compared to the greedy strategy for interception based redirection. Additionally in large networks, if redirection is based on individual client IP addresses, our heuristics reduce the number of caches by 17% compared to the case where redirection is based on entire IP prefix ranges.

In future work we intend to verify our algorithm by implementing it within a large VPN and to evaluate the impact of source routing, which is available in MPLS based VPNs, on the number of caches. The potential benefit of source routing is that if a client is redirected to a particular cache the links used to transfer the data are not predetermined by shortest path. They can rather be optimized by the redirection system as to avoid bottleneck links. This has the potential to reduce the number of caches even further.

7. REFERENCES

[1] GT-ITM: Georgia Tech Internetwork Topology Models. `http://www.cc.gatech.edu/projects/gtitm/`.

[2] A. Barbir, B. Cain, F. Douglis, M. Green, M. Hofmann, R. Nair, D. Potter, and O. Spatscheck. Known CN Request-Routing Mechanisms . `http://www.ietf.org/internet-drafts/draft-ietf-cdi-known-request-routin% g-01.txt`.

[3] M Cieslak, D Forster, G Tiwana, and R Wilson. Web Cache Coordination Protocol V2.0. `http://www.cisco.com/warp/public/732/Tech/switching/wccp/docs/draft-wil%son-wrec-wccp-v2-00.rtf`.

[4] Thomas H. Cormen, Charlse E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[5] Michael R. Garey and David S. Johnson. *Computers and Intractability*. W.H. Freeman and Company, 1979.

[6] Sugih Jamin, Cheng Jin, Anthony Kurc, Yuval Shavitt, and Danny Raz. Constrained Mirror Placement on the Internet. In *Proceedings of IEEE Infocom*, April 2001.

[7] J. Kangasharju, J. W. Roberts, and K. W. Ross. Object replication strategies in content distribution networks. In *6th International Web Content Caching and Distribution Workshop*, June 2001.

[8] Magnus Karlsson, Christos Karamaolis, and Mallik Mahalingam. A framework for evaluating replica placement algorithms. `http://www.hpl.hp.com/personal/Magnus_Karlsson/papers/rp_framework.pdf`, 2002. Submitted for publication.

[9] Magnus Karlsson and Mallik Mahalingam. Do We Need Replica Placement Algorithms in Content Delivery Networks? In *Proceeding of 7th International Web Content Caching and Distribution Workshop*, August 2002.

[10] Donald E. Knuth. *The Stanford GraphBase*. Addison-Wesley, 1993.

[11] Lili Qiu, Venkata N. Padmanabhan, and Geoffrey M. Voelker. On the Placement of Web Server Replicas. In *Proceedings of INFOCOM*, April 2001.

[12] Sherlia Shi and Jonathan Turner. Placing servers in overlay networks. In *Symposium on Perform ance Evaluation of Computer and Telecommunication Systems (SPETS)*, July 2002.