

# Proxy Placement for Server-based Multicast Overlays

Min-You Wu, Yan Zhu, and Wei Shu  
Department of Electrical and Computer Engineering  
The University of New Mexico

*Abstract* — A study on the proxy placement for server-based multicast is presented in this paper. Server-based multicast is an approach implementing the multicast function with proxy servers. It has many advantages compared to the IP multicast. Many routing algorithms have been proposed to build an overlay network when proxy locations have been determined. On the other hand, the problem of where to place proxies has not been extensively studied yet. The goals of proxy placement can be minimization of delay time, minimization of bandwidth consumption, or minimization of both. These performance metrics have been studied in this paper and a number of algorithms are proposed for multicast proxy placement. A performance study is also presented.

## 1 Introduction

Multicast is an important technique that can distribute popular contents to many users with minimal bandwidth consumption. Multicast is able to remove the hot spots from networks when many users access the same content, especially live contents. Multicast is also used to distribute popular data to many users. IP multicast has been researched and developed for more than a decade [8, 9], but it has not been widely deployed [11]. As an alternative to the IP multicasting, the rapid development of server-based multicasting technologies has increased greatly over the last couple of years [5, 6, 14, 26, 22]. Server-based multicast challenges the traditional IP multicast by moving up the multicast support from the network layer onto the application layer based on only unicast IP service. It potentially provides better solutions to scalability, flexibility, heterogeneous support, and efficiency for various enabling applications. Server-based multicast can also work across both space and time by using proxy servers as caches [30].

A general approach to implement server-based multicast is to build an overlay network. Many routing algorithms have been proposed for multicast tree building [5, 6, 14, 12, 22, 13]. Some of them proposed to minimize the delay time between the source and the clients [5]; others proposed to minimize the bandwidth consumption [6]. Also, it has been proposed that the bandwidth between the original server and the end-user should not be sacrificed when minimizing the bandwidth consumption [14]. In fact, delay time and bandwidth consumption are both important considerations for multicast. Bandwidth consumption is the prime concern when multicast is used. On the other hand, delay should not increase too much since multicast is often used for real-time applications. Balance between these two metrics is essential for the

best design of the server-based multicast. Comparison studies show that the shortest-path tree generates the smallest delay, Steiner minimal tree generates the lowest bandwidth consumption, and the core-based tree falls in between the two extremes [28, 27, 10]. However, no metric exists that includes both criteria, delay and bandwidth consumption. In this paper, we will define a hybrid optimal tree that optimizes their trade-off.

The routing algorithms assume that the locations of proxies are known and fixed. The problem of where to place these proxies has not been extensively studied yet. In fact, the proxy placement is much more important than the routing problem. Without an advanced placement method, routing cannot be effective. Currently, the cache/mirror/CDN proxy/replica placement problem [21, 23, 20, 16, 15] has been extensively studied, however, few addresses the *multicast proxy placement (MPP)* problem. Despite the fact that the multicast proxy placement and the cache proxy placement share some similarities, the two problems have different assumptions, so the cache proxy placement technique is not applicable to the MPP problem. The cache proxy placement problem assumes that contents are cached in the cache proxies and can be accessed at different times. The MPP problem assumes that the content is distributed to many users at the same time. The delay time in a cache network depends on the distance between the client and its nearest cache proxy that stores the desired content; whereas, in a multicast network, it depends on the distance between the client and the source.

In this paper, our goal is to develop the methodology and algorithms for multicast proxy placement. In particular, we formulate the MPP problem and its performance metrics, as well as its association with the *multicast proxy routing (MPR)* problem. We introduce a number of multicast proxy placement algorithms for optimal proxy placement. Experiments are conducted for the performance of MPP algorithms, which increases our understanding of the MPP problem. Guidelines are given for general proxy placement. The results obtained from this work are also applicable to Mbone router placement.

The remainder of this paper is organized as follows. Section 2 formulates the MPR and MPP problems and introduces the performance metrics. An example illustrates why proxy placement is important. Underlying routing algorithms are presented in Section 3. Section 4 presents four groups of MPP algorithms: routing-independent, routing-aware, greedy, and local search. Experimental results are presented in Section 5. Section 6 discusses related works, and Section 7 concludes the paper.

## 2 Problem Formulation

The server-based multicast is implemented as an overlay network. An overlay network consists of a set of proxies placed on a substrate network. The substrate network is a collection of routers connected by links. Then a multicast tree of proxies is built with a routing algorithm. The multicast tree can be single-sourced

or multi-sourced. In this paper, we only consider the single-sourced multicast tree.

From the perspective of a client, the delay time is the main concern. On the other hand, the cost in terms of total bandwidth consumption is a main indication of network efficiency from a network provider's point of view. Thus, the quality of a multicast tree can be judged by two measures: the delay and the cost. The delay of a multicast tree is evaluated in terms of the end-to-end delay between the source and the client. The cost of a multicast tree is the sum of bandwidth consumption of all links. Bandwidth consumption measures the usage of network resources. The more hops a message travels, the more resources that are consumed.

We first present a model of the substrate network and the overlay network before the objective functions are defined. The substrate network is represented by an undirected graph  $G$ , consisting of a set of  $N$  nodes  $\mathcal{S}=\{n_1, n_2, \dots, n_N\}$ . Each node has a weight  $w(i)$  to represent the number of clients concentrated at node  $n_i$ . There is one source node,  $n_s \in \mathcal{S}$ . The distance of link  $l_{i,j}$  of the substrate network between nodes  $n_i$  and  $n_j$  is denoted as

$$e(i, j) = e(j, i) = \begin{cases} \infty & \text{if there is no link } l_{i,j} \text{ between nodes } n_i \text{ and } n_j \\ r(l_{i,j}) & \text{otherwise, } r(l_{i,j}) \text{ is a finite constant preassigned to link } l_{i,j} \end{cases}$$

The parameter  $r(l_{i,j})$  can be interpreted as delay, cost, hop count, etc.

A multicast proxy is a duplication engine that is able to duplicate the incoming data stream for multiple output links. A multicast overlay network consists of a set of  $P$  proxies  $\{p_1, p_2, \dots, p_P\}$  plus the source node  $n_s$ . These proxies are to be placed to  $P$  nodes. All nodes with proxies placed plus the source node  $n_s$  are called the *multicast nodes*. Thus, there are  $P + 1$  multicast nodes with the duplication capability. A routing algorithm connects the multicast nodes onto a multicast tree based on some criteria, such as minimal delay, minimal bandwidth consumption, or a combination of them. A non-multicast node is routed to a proxy according to the same criteria. More specifically, a multicast tree  $M$  can be defined by  $\{X, P, Y\}$ . Here,  $X$  is a mapping function of proxy placement

$$x(i) = \begin{cases} 1 & \text{if } n_i \equiv n_s \text{ or a proxy is placed at } n_i \\ 0 & \text{otherwise} \end{cases}$$

$P$  is a function for each node to know its immediate parent multicast node in the tree:

$$p(i) = k \text{ if } x(k) \equiv 1 \text{ and } n_k \text{ delivers the stream to } n_i \text{ by using a unicast connection}$$

and  $Y$  defines which real links are used when  $n_{p(i)}$  delivers the stream to  $n_i$  by using a unicast connection,

$$y(i, u, v) = \begin{cases} 1 & \text{if } l_{u,v} \text{ is used for path from } n_{p(i)} \text{ to } n_i \\ 0 & \text{otherwise} \end{cases}$$

First, we define two basic metrics, the total delay and the total bandwidth consumption. For each node  $n_i$ ,

the end-to-end delay from the source node  $n_s$  to  $n_i$  is

$$d_M(i) = w(i) \cdot (d' + d'') = w(i) \cdot \left( \overbrace{\sum_{u=1}^N \sum_{v=1}^N e(u, v) \cdot y(i, u, v)}^{d'} + \underbrace{d_M(p(i))}_{d''} \right) \quad (1)$$

where,  $d'$  is the delay from multicast node  $n_{p(i)}$  to node  $n_i$ , and  $d''$  is the delay from source node  $n_s$  to  $n_{p(i)}$ . The total delay of a multicast tree is the sum of delays between the source and every clients along the multicast tree  $M$ .

$$D_M = \sum_{i=1}^N d_M(i) \quad (2)$$

For node  $n_i$ , the bandwidth consumption of delivering one stream from its parent multicast node  $n_{p(i)}$  is

$$c_M(i) = \sum_{u=1}^N \sum_{v=1}^N e(u, v) \cdot y(i, u, v). \quad (3)$$

To calculate the bandwidth consumption for a non-multicast node  $n_i$ ,  $w(i)$  streams are duplicated from its parent multicast node  $n_{p(i)}$ , where  $w(i)$  is the number of clients concentrated at node  $n_i$ . For a multicast node  $n_i$ , only one stream is sent from its parent multicast node  $n_{p(i)}$ . We use  $b_M$  to compute the bandwidth consumption,

$$b'_M(i) = \begin{cases} c_M(i) & \text{if } x(i) \equiv 1 \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad b''_M(i) = \begin{cases} w(i) \cdot c_M(i) & \text{if } x(i) \equiv 0 \\ 0 & \text{otherwise} \end{cases}. \quad (4)$$

The total bandwidth consumption of multicast data distribution is defined as

$$B_M = \sum_{i=1}^N b_M(i) = \sum_{i=1}^N b'_M(i) + \sum_{i=1}^N b''_M(i) \quad (5)$$

Note that  $e(u, v)$  used in Equations 1 and 3 can be replaced with different variables as  $e_1(u, v)$  and  $e_2(u, v)$ , where  $e_1(u, v)$  means the delay of link  $l_{u,v}$  and  $e_2(u, v)$  means the bandwidth consumption via link  $l_{u,v}$ . For simplicity, we treat them the same in this paper.

Both  $D_M$  and  $B_M$  depend on the routing algorithm used to construct the multicast tree  $M$ . Achieving minimal delay and reducing bandwidth consumption are usually contradictory to each other. It has been known that the Minimal Spanning Tree (MST) minimizes the bandwidth consumption, but involves a longer delay than that of the Shortest Path Tree (SPT). On the other hand, SPT minimizes the delay, but consumes more bandwidth than the MST routing. In order to optimize the both metrics simultaneously, we define a new metric, the delay-bandwidth-consumption  $DB_M$  as follows

$$DB_M = D_M \times B_M \quad (6)$$

Here, the product of  $D_M$  and  $B_M$  is used to construct the hybrid metric  $DB_M$  instead of a summation of the two, since  $D_M$  and  $B_M$  are different units and their values are difficult to compare. In the situation where either one is to be emphasized more than the other, we can raise its impact by utilizing a weighted power.

In general, a multicast tree  $M$  defined by  $\{X, P, Y\}$  is relevant to both the proxy placement and routing as described in the following table:

$M = \{X, P, Y\}$	
Proxy Placement	$X$
Routing	$P$ and $Y$

Therefore, the problem can be formulated as

- **PROBLEM 1.** Given the proxy placement  $X$ , design a routing algorithm  $A_R$  to construct the multicast tree,  $P$  and  $Y$ , subject to minimizing  $D_M$ ,  $B_M$ , or  $DB_M$
- **PROBLEM 2.** By limiting the number of proxies,  $P = \sum_{i=1}^N x(i) - 1$ , to be placed, design a placement algorithm  $A_p$  to generate a mapping function  $X$ , subject to optimizing  $D_M$ ,  $B_M$ , or  $DB_M$  with a routing algorithm  $A_R$

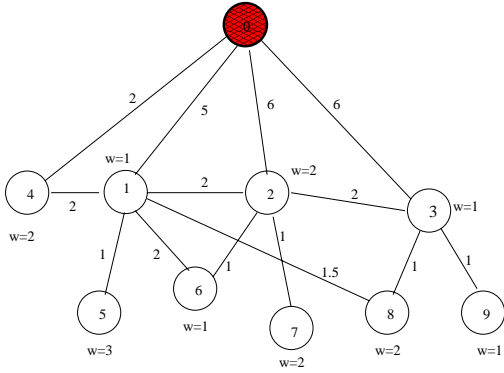
Two design problems have been formulated for a multicast overlay network, the multicast proxy routing (MPR) algorithm and the multicast proxy placement (MPP) algorithm, together with three important performance metrics,  $D_M$ ,  $B_M$ , and  $DB_M$ . Solving PROBLEM 2 usually is more difficult than PROBLEM 1. Based on performance metrics to be targeted, an optimal solution to PROBLEM 2 may inevitably fall into the class of NP-complete problems. As an example, should there exist a polynomial solution to PROBLEM 2 with optimal  $B_M$  as its objective function, it would be reducible to the well-known *minimum k-median* problem that had been proved to be NP-complete [17]. In some cases, preference can be given to the co-design between PROBLEM 1 and PROBLEM 2, which is called the routing-aware placement algorithm in this paper.

To illustrate both PROBLEM 1 and PROBLEM 2, a small sample graph  $G_{\text{sample1}}$  is given in Figure 1 with  $N = 10$  and  $n_s = 0$ .

**Example 1.a.** For PROBLEM 1, a proxy placement  $X_1$  is given as

$i$	0	1	2	3	4	5	6	7	8	9
$x_1(i)$	1	1	1	1	0	0	0	0	0	0

Assume that three different routing algorithms,  $A_{R_1}$ ,  $A_{R_2}$ , and  $A_{R_3}$ , are applied. The resultant multicast trees are shown in Table 1. as well as their corresponding metrics  $D_M$ ,  $B_M$ , and  $DB_M$ . Among them  $M_1$  and  $M_2$  are graphically illustrated in Figure 2. As the result clearly indicates, algorithm  $A_{R_1}$  minimizes  $D_M$ , algorithm  $A_{R_2}$  minimizes  $B_M$ , and algorithm  $A_{R_3}$  minimizes  $DB_M$ . To target PROBLEM 1, for a given



$n_i$	$w(i)$	$e(i,0)$	$e(i,1)$	$e(i,2)$	$e(i,3)$	$e(i,4)$	$e(i,5)$	$e(i,6)$	$e(i,7)$	$e(i,8)$	$e(i,9)$
$n_0$	1	-	5	6	6	2	-	-	-	-	-
$n_1$	1	5	-	2	-	2	1	2	-	1.5	-
$n_2$	2	6	2	-	2	-	-	1	1	-	-
$n_3$	1	6	-	2	-	-	-	-	-	1	1
$n_4$	2	2	2	-	-	-	-	-	-	-	-
$n_5$	3	-	1	-	-	-	-	-	-	-	-
$n_6$	1	-	2	1	-	-	-	-	-	-	-
$n_7$	2	-	-	1	-	-	-	-	-	-	-
$n_8$	2	-	1.5	-	1	-	-	-	-	-	-
$n_9$	1	-	-	-	1	-	-	-	-	-	-

Figure 1:  $n_i$ ,  $w(i)$ , and  $e(i, j)$  of a sample graph  $G_{\text{sample1}}$

	$i$	0	1	2	3	4	5	6	7	8	9	$D_M$	$B_M$	$DB_M$
$M_1$	$p_1$	-	0	1	0	0	1	1	2	1	3			
	$y_1$	-	$l_{0,4}, l_{4,1}$	$l_{1,2}$	$l_{0,3}$	$l_{0,4}$	$l_{1,5}$	$l_{1,6}$	$l_{2,7}$	$l_{1,8}$	$l_{3,9}$			
	$d_1$	-	4	12	6	4	15	6	12	11	7	79	27	2133
	$b_1$	-	4	2	6	4	3	2	2	2	3			
$M_2$	$p_2$	-	0	1	2	0	1	2	2	3	3			
	$y_2$	-	$l_{0,4}, l_{4,1}$	$l_{1,2}$	$l_{2,3}$	$l_{0,4}$	$l_{1,5}$	$l_{2,6}$	$l_{2,7}$	$l_{3,8}$	$l_{3,9}$			
	$d_2$	-	4	12	8	4	15	7	14	18	9	91	21	1911
	$b_2$	-	4	2	2	4	3	1	2	2	1			
$M_3$	$p_3$	-	0	1	2	0	1	2	2	1	3			
	$y_3$	-	$l_{0,4}, l_{4,1}$	$l_{1,2}$	$l_{2,3}$	$l_{0,4}$	$l_{1,5}$	$l_{2,6}$	$l_{2,7}$	$l_{1,8}$	$l_{3,9}$			
	$d_3$	-	4	12	8	4	15	7	14	11	9	84	22	1848
	$b_3$	-	4	2	2	4	3	1	2	3	1			

Table 1:  $M_1$ ,  $M_2$ , and  $M_3$  of  $G_{\text{sample1}}$  with  $X_1 = \{n_0, n_1, n_2, n_3\}$

placement  $X_1$ , based on metrics used for optimization, a routing algorithm can be selected to construct a multicast tree. For example, if the delay is a crucial issue for a particular application, routing algorithm  $A_{R1}$  should be applied.

**Example 1.b.** For PROBLEM 2, the number of proxies,  $P$ , is limited to 3; various placement algorithms can be used to generate different  $X_M$ . Assume a placement algorithm  $A_{P1}$  results in  $X_1$  shown in Example 1.a; another placement algorithm  $A_{P2}$  results in a different placement  $X_2$  as shown below,

$i$	0	1	2	3	4	5	6	7	8	9
$x_2(i)$	1	1	0	1	0	0	1	0	0	0

Again, apply routing algorithms  $A_{R1}$ ,  $A_{R2}$ , and  $A_{R3}$  onto placement  $X_2$ . The resultant multicast trees are shown in Table 2. Among them, the resultant multicast trees,  $M_3$  and  $M_6$ , from two placement algorithms,

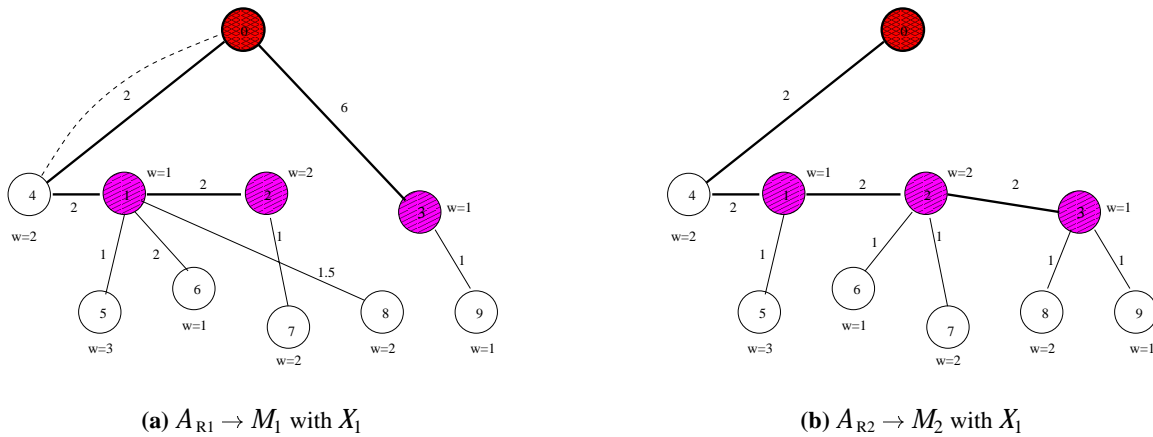


Figure 2: Multicast tree routing for  $G_{\text{sample1}}$

$A_{P1}$  and  $A_{P2}$ , with the same routing algorithm  $A_{R3}$  are shown in Figure 3. In order to compare two placement algorithms,  $A_{P1}$  and  $A_{P2}$ , the metrics of the six different multicast trees are summarized in Table 3. Obviously, placement algorithm  $A_{P1}$  is a better choice compared to  $A_{P2}$ . In general, generation

	$i$	0	1	2	3	4	5	6	7	8	9	$D_M$	$B_M$	$DB_M$
$M_4$	$p_1$	-	0	1	0	0	1	1	1	1	3			
	$y_1$	-	$l_{0,4}, l_{4,1}$	$l_{1,2}$	$l_{0,3}$	$l_{0,4}$	$l_{1,5}$	$l_{1,6}$	$l_{1,2}, l_{2,7}$	$l_{1,8}$	$l_{3,9}$			
	$d_1$	-	4	12	6	4	15	6	14	11	7	79	33	2607
	$b_1$	-	4	4	6	4	3	2	6	3	1			
$M_5$	$p_2$	-	0	1	1	0	1	1	1	3	3			
	$y_2$	-	$l_{0,4}, l_{4,1}$	$l_{1,2}$	$l_{1,2}, l_{2,3}$	$l_{0,4}$	$l_{1,5}$	$l_{1,6}$	$l_{1,2}, l_{2,7}$	$l_{1,8}$	$l_{3,9}$			
	$d_2$	-	4	12	6.5	4	15	6	14	15	7.5	84	28.5	2394
	$b_2$	-	4	4	2.5	4	3	2	6	2	1			
$M_6$	$p_3$	-	0	1	1	0	1	1	1	1	3			
	$y_3$	-	$l_{0,4}, l_{4,1}$	$l_{1,2}$	$l_{1,2}, l_{2,3}$	$l_{0,4}$	$l_{1,5}$	$l_{2,6}$	$l_{1,2}, l_{2,7}$	$l_{1,8}$	$l_{3,9}$			
	$d_3$	-	4	12	6.5	4	15	6	14	11	7.5	80	29.5	2360
	$b_3$	-	4	4	2.5	4	3	2	6	3	1			

Table 2:  $M_4$ ,  $M_5$ , and  $M_6$  of  $G_{\text{sample1}}$  with  $X_2 = \{n_0, n_1, n_3, n_6\}$

of a good multicast tree demands both appropriate placement and routing algorithms. Routing algorithms play an important role, but without an appropriate placement, routing algorithms alone will not be able to produce good performance.

	$A_{P1} \Rightarrow X_1$			$A_{P2} \Rightarrow X_2$		
	$A_{R1} \rightarrow M_1$	$A_{R2} \rightarrow M_2$	$A_{R3} \rightarrow M_3$	$A_{R1} \rightarrow M_4$	$A_{R2} \rightarrow M_5$	$A_{R3} \rightarrow M_6$
$D_M$	79	91	84	79	84	80
$B_M$	27	21	22	33	28.5	29.5
$DB_M$	2133	1911	1848	2607	2394	2360

Table 3:  $D_M$ ,  $B_M$ , and  $DB_M$  of  $G_{\text{sample1}}$  with  $X_1$  and  $X_2$

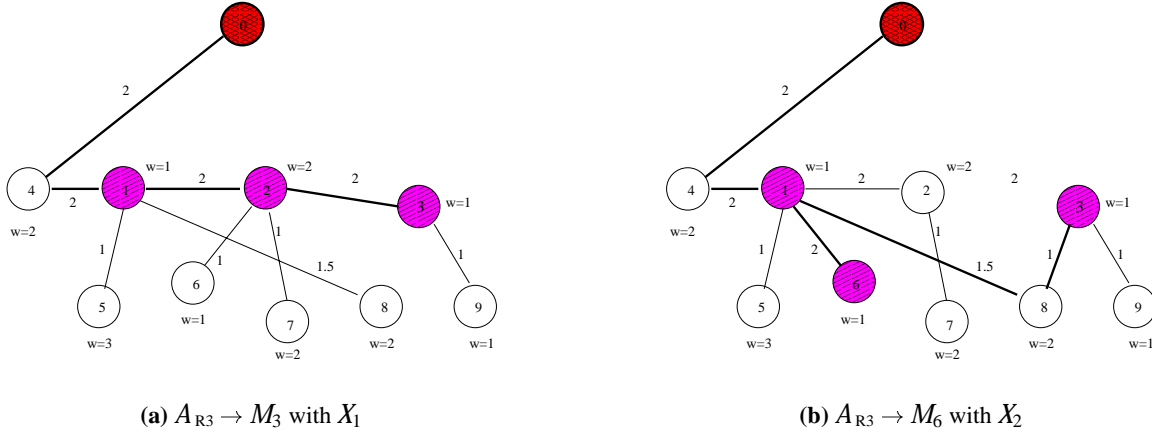


Figure 3: Multicast proxy placement for  $G_{\text{sample1}}$

### 3 Underlying Routing Algorithms

Given a placement  $X$ , a routing algorithm  $A_R$  will build a multicast tree  $M$  optimized toward certain metrics. Based on the targeted metrics, these multicast trees can be classified into the following three categories.

#### 3.1 Shortest Path Tree (SPT) — $M_{\text{SPT}}$ minimizing $D_M$

In a  $M_{\text{SPT}}$ , every node  $n_i \in S$  connects to the source  $n_s$  through the shortest path. If the shortest path of a node to the source includes some proxies, the node is connected to the nearest proxy along the path; otherwise it is connected to the source directly.  $M_{\text{SPT}}$  minimizes the delay  $d_M(i)$  from  $n_s$  to every node  $n_i$ , respectively, which turns out to optimize the total delay  $D_M$ . Thus, regardless of placement  $X$ , the metric  $D_M$  remains the same since the shortest paths for every node will not change. Figure 4 shows the routing algorithm  $A_{\text{SPT}}$ . In step 1, We use the Dijkstra's algorithm to construct a single-source  $M_{\text{SPT}}$  and retain its complexity as  $O(N^2)$ . In step 2, the  $M_{\text{SPT}}$  tree has been traversed to establish  $P_M$  and  $Y_M$ . Since every link on the  $M_{\text{SPT}}$  tree will be visited at most  $N$  times and the number links on  $M_{\text{SPT}}$  is in order of  $N$ , the time

---

**Function:** given  $X$ , generate  $P_M$  and  $Y_M$

**Objective:** minimize  $D_M$

- step 1) Apply the Dijkstra's algorithm to the entire graph  $G$  to generate a single-source SPT with:  
 $sp(i)$  representing the shortest path between  $n_i$  and  $n_s$   
 $nextNodeOnSPT(i)$  representing the next node on the shortest path towards  $n_s$
- step 2) for each node  $n_i$  excluding  $n_s$ , let  $k_1 = i$   
do  $k_2 = nextNodeOnSPT(k_1)$ ;  
 $y(i, k_1, k_2) = 1$ ;  
 $k_1 = k_2$ ;  
while  $(x(k_2) \neq 1)$   
 $p(i) = k_2$ ;  
 $p(s) = s$ ;
- 

Figure 4: Routing Algorithm  $A_{SPT}$  for  $M_{SPT}$  minimizing  $D_M$

spent in step 2 is within  $O(N^2)$  too.

### 3.2 Minimal Spanning Tree (MST) — $M_{MST}$ minimizing $B_M$

The Minimal Spanning Tree (MST) is a minimal-cost tree connecting every node in a graph. However, the fact that there exist some non-multicast nodes changes the presumption that a MST can always lead to minimal cost. The objective here is to minimize  $B_M$ , which consists of two parts, as shown in Equations 4 and 5:

- $b'_M(i)$  for all multicast nodes with  $x(i) = 1$
- $b''_M(i)$  for all non-multicast nodes with  $x(i) = 0$

Figure 5 shows the routing algorithm  $A_{MST}$ . In order to minimize  $B_M$ , a subgraph with all multicast nodes is constructed by utilizing the shortest path among them. In step 1, for simplicity, an all-pair SPT is constructed with a complexity of  $O(N^3)$ . In reality, step 2 needs all-pair SPT among all multicast nodes and step 5 needs  $P$  single-pair SPT, including all non-multicast nodes, sourced at each multicast node. Step 2 constructs the subgraph and step 3 applies the Prim's algorithm on this subgraph to find the MST with a complexity of  $O(P^2)$  [1]. In this way, the first part,  $b'_M(i)$ , can be minimized. The commonly used Steiner Minimal Tree (SMT) does not work in this case since any included non-multicast nodes cannot replicate the traffic. In step 4, the MST tree has been traversed to establish  $P_M$  and  $Y_M$  for all multicast nodes. In step 5, to minimize the second part,  $b''_M(i)$ , all non-multicast nodes are directed to the nearest multicast nodes, which guarantees leading to optimal  $b''_M(i)$ . Among all five steps in Figure 5, step 1 has the dominate complexity, which determines the complexity of the routing algorithm as  $O(N^3)$ .

---

**Function:** given  $X$ , generate  $P_M$  and  $Y_M$

**Objective:** minimize  $B_M$

- step 1) apply the Dijkstra's algorithm to the entire graph  $G$  to generate all-pair SPTs with:  
     $sp(i, j) = sp(j, i)$  representing the shortest path between  $n_i$  and  $n_j$   
    nextNodeOnSPT( $i, j$ ) representing the next node on the shortest path toward  $n_j$
- step 2) construct a subgraph  $O$  with  
    all nodes with  $x(i) = 1$   
    use  $sp(i, j)$  as the virtual link cost  $v(i, j)$  between  $n_i$  and  $n_j$ , where  $x(i) = x(j) = 1$
- step 3) apply the Prim's algorithm to the subgraph  $O$  to generate a MST with  
     $p(i) = j$  if  $v(i, j)$  is included in MST towards  $n_s$   
    nextNodeOnMST( $i$ ) representing the next node on the path in MST towards  $n_s$
- step 4) for each node  $n_i$  with  $x(i) = 1$  excluding  $n_s$ , let  $k_1 = i$   
    do  $k_2 = \text{nextNodeOnMST}(k_1)$ ;  
         $y(i, k_1, k_2) = 1$ ;  
         $k_1 = k_2$ ;  
    while ( $x(k_2) \neq 1$ )  
     $p(s) = s$ ;
- step 5) for each node  $n_i$  with  $x(i) = 0$ , let  $k_1 = i$   
    let  $p(i) = q$  be a multicast node such that  $sp(i, q) \leq sp(i, j)$ , where  $x(q) = x(j) = 1$   
    do  $k_2 = \text{nextNodeOnSPT}(k_1, q)$ ;  
         $y(i, k_1, k_2) = 1$ ;  
         $k_1 = k_2$ ;  
    while ( $k_2 \neq q$ )
- 

Figure 5: Routing Algorithm  $A_{\text{MST}}$  for  $M_{\text{MST}}$  minimizing  $B_M$

### 3.3 Hybrid Optimal Tree (HOT) — $M_{\text{HOT}}$ minimizing $DB_M$

The *Hybrid Optimal Tree (HOT)* minimizes  $DB_M$ , a compromise between SPT and MST. Many applications can promote this newly defined metric  $DB_M$ . For example, realtime video multicast streaming can tolerate neither long delay nor high bandwidth consumption. Two algorithms are used here to generate the multicast tree  $M_{\text{HOT}}$ . The first one is the Prim-Dijkstra's algorithm.

**Prim-Dijkstra's (P-D) algorithm  $A_{\text{P-D}}$**  — The Prim-Dijkstra's algorithm [18] can be used to find the  $M_{\text{HOT}}$  tree. It builds a tree by starting at a node and bringing in nodes by picking the one with the best label:  $MIN_k(\alpha \times \text{dist}(n_s, p_k) + \text{dist}(p_k, n_i))$ , where  $0 \leq \alpha \leq 1$  is a constant used to parameterize the calculation. The complexity of Prim-Dijkstra's algorithm  $A_{\text{P-D}}$  is  $O(N^2)$  and details are described in [18].

Since the P-D algorithm  $A_{\text{P-D}}$  is unable to guarantee a minimal  $DB_M$  value and a parameter needs to be adjusted, we use a greedy algorithm to directly minimize the  $DB_M$  value.

**Direct-DB (DDB) algorithm**  $A_{\text{DDB}}$  — DDB routes proxies one by one as shown in Figure 6, starting from the nearest proxy to the source. The first proxy connects to the source through the shortest path. The following proxies connect to the source or a placed proxy, whichever minimizes  $DB_M$ . After the multicast tree is established, each non-multicast node connects to the multicast node that generates the smallest  $DB_M$  value. The complexity of this algorithm is  $O(PN^2)$ . This algorithm generates a tree that balances delay and cost requirements.

---

**Function:** given  $X$ , generate  $P_M$  and  $Y_M$

**Objective:** minimize  $DB_M$

- step 1) apply the Dijkstra's algorithm to the entire graph  $G$  to generate all-pair SPTs with:
    - $sp(i, j) = sp(j, i)$  representing the shortest path between  $n_i$  and  $n_j$
    - $\text{nextNodeOnSPT}(i, j)$  representing the next node on the shortest path toward  $n_j$
  - step 2) sort all multicast nodes in ascending order of length of their shortest path towards  $n_s$ .  
reset  $\text{mark}[i] = \text{false}$  for all  $n_i$  and  $\text{mark}[s] = \text{true}$
  - step 3) establish an init routing with all  $n_i$  following the SPT towards  $n_s$  and compute the init value  $DB_{\text{cur}}$
  - step 4) for each multicast node  $n_i$  with  $x(i) = 1$ ,  $\text{mark}[i] = \text{false}$ , according to their order sorted by step 2
    - let  $p(i) = q$  be an already-routed multicast node such that  $\text{gain}(i, q) \geq \text{gain}(i, j)$ ,  
where  $\text{mark}[q] = \text{mark}[j] = \text{true}$
    - $\text{gain}(i, j) = DB_{\text{cur}} - DB_{\text{new}}$ , let  $k_1 = i$
    - do  $k_2 = \text{nextNodeOnSPT}(k_1, q)$ ;
    - $y(i, k_1, k_2) = 1$ ;
    - $k_1 = k_2$ ;
    - while  $(k_2 \neq q)$
    - update  $DB_{\text{cur}}$
    - $p(s) = s$ ;
  - step 5) for each node  $n_i$  with  $x(i) = 0$ , let  $k_1 = i$ 
    - let  $p(i) = q$  be a multicast node such that  $sp(i, q) \leq sp(i, j)$ , where  $x(q) = x(j) = 1$
    - do  $k_2 = \text{nextNodeOnSPT}(k_1, q)$ ;
    - $y(i, k_1, k_2) = 1$ ;
    - $k_1 = k_2$ ;
    - while  $(k_2 \neq q)$
- 

Figure 6: Routing Algorithm  $A_{\text{DDB}}$  for  $M_{\text{HOT}}$  minimizing  $DB_M$

In summary, the design of routing algorithms has been motivated by various performance metrics. Depending on characteristics of applications, the selection of the right routing algorithm is crucial to overall success in performance. Usually, routing algorithms are applied, after the placement of proxies is known, to construct the multicast tree. Sometimes, routing algorithms can be used to help the decision of proxy placement, as discussed in the following section. In any case, construction of multicast tree cannot be completed without good design and implementation of routing algorithms. Building synergy between routing and placement is even more important.

## 4 Proxy Placement Algorithms

A MPP algorithm finds the optimal locations for proxies. The objective functions can be minimizing  $D_M$ ,  $B_M$ , or  $DB_M$ . Here, the placement algorithms are presented in four categories: routing-independent, routing-aware, greedy, and local search.

### 4.1 Routing-independent algorithms

The routing-independent placement algorithms do not rely on routing information. Routing is performed only after the proxies are placed. Four algorithms are presented here. Random and Max-degree placement algorithms are for arbitrary topology, and Strategic and Advanced Strategic Placement (ASP) algorithms are for the Internet topology.

**Random.** The random algorithm randomly chooses  $P$  locations among  $N - 1$  nodes (excluding  $n_s$ ) to place proxies. Its complexity is  $O(N)$ .

**Max-Degree.** The max-degree algorithm places  $P$  proxies to the nodes with the highest node degrees. The complexity of this algorithm is  $O(N \log N)$  due to sorting of nodes.

**Strategic.** The strategic placement selects strategically important nodes to place the proxies. In the Internet topology, proxies are placed to the transit nodes first, then the stub nodes as described in [14]. Ties are broken randomly. The complexity of this algorithm is  $O(N)$ .

**Advanced Strategic Placement (ASP).** The ASP algorithm is also for the Internet topology. First, we select the transit nodes in the order of the number of stub nodes connected to it. Then, we select the stub nodes that are connected to transit nodes directly in the order of the number of stub nodes connected to it. Finally, other stub nodes are selected in the order of the node degree. The complexity of this algorithm is  $O(N \log N)$ .

### 4.2 Routing-aware algorithms

Routing-aware placement algorithms find a better placement by utilizing the routing information. First, a routing algorithm  $A_R$  is applied to build a multicast tree  $M_{\text{INT}}$ , by assuming all nodes are multicast nodes. Then, placement is performed based on this multicast tree. Two routing-aware algorithms are presented, Max-Traffic and Max-Traffic-Distance (Max-TD). We use  $O_{\text{routing}}$  to represent the complexity of the routing algorithm to build the multicast tree.

For a given multicast tree  $M_{\text{INT}}$ , traffic  $T$  of a node is defined as traffic passing through the node plus the traffic terminated at the node by assuming the source node is only one multicast node. If  $T$  of a node is more than 1, the node is a *branching node*. The number of branching nodes  $Q \leq N - 1$ . If the number of proxies to be placed  $P = Q$ , there is no repeated traffic on every link, and the bandwidth consumption is minimal. When  $P < Q$ , the following algorithms select nodes to place  $P$  proxies.

---

**Function:** for a routing algorithm  $A_R$ , generate placement  $X$   
**Placement priority:** nodes with Max-Traffic

step 1) Let  $x(i) = 1$  for all  $i$ , apply  $A_R$  to generate a  $M_{\text{INT}}$  with  
 $\text{nextNodeOnTree}(i, p(i))$  representing the path towards  $n_{p(i)}$

step 2) Let  $t(i) = 0$  for all  $i$

step 3) For each  $n_i$ , let  $k_1 = i, k_3 = p(i)$  and  $t(i) = t(i) + w(i)$   
do  $k_2 = \text{nextNodeOnTree}(k_1, k_3)$   
 $t(k_2) = t(k_2) + w(i)$   
 $k_1 = k_2$ , if  $k_2 = k_3, k_3 = p(k_3)$   
while  $(k_2 \neq s)$

step 4) Sort all nodes in ascending order of  $t(i)$

step 5) Let  $x(i) = 0$  for all  $i$ , and  $x(s) = 1$   
For the first  $P$  node in order let  $x(i) = 1$

---

Figure 7: Placement Algorithm: Max-Traffic

---

**Function:** for a routing algorithm  $A_R$ , generate placement  $X$   
**Placement priority:** nodes with Max-TD

step 1) Let  $x(i) = 1$  for all  $i$ , apply  $A_R$  to generate a  $M_{\text{INT}}$  with  
 $\text{nextNodeOnTree}(i, p(i))$  representing the path towards  $n_{p(i)}$

step 2) Let  $x(i) = 0$  for all  $i$ , and  $x(s) = 1$

step 3) For  $k = 0$  to  $P$ , let  $t(i) = 0$  for all  $i$

step 4) For each  $n_i$ , let  $k_2 = i$   
do  $k_2 = \text{nextNodeOnTree}(k_1, p(i))$   
 $t(k_2) = t(k_2) + 1$  if  $x(i) = 1$   
 $t(k_2) = t(k_2) + w(i)$  if  $x(i) = 0$   
 $k_1 = k_2$   
while  $(k_2 \neq p(i))$

step 5) find a node  $j$  such that  $t(j) * d(j) \geq t(u) * d(u)$ , where  $x(j) = x(u) = 0$

step 6) let  $x(j) = 1$

---

Figure 8: Placement Algorithm: Max-TD

**Max-Traffic.** The traffic is computed for each node based on routing in  $M_{\text{INIT}}$  as shown in Figure 7.

The proxies are placed at the top  $P$  locations with the most concentrated traffic. Its complexity is  $O(N \log N) + O_{\text{routing}}$ .

**Max-Traffic-Distance (Max-TD).**  $TD$  is the product of the node traffic and the distance between the node  $n_i$  and its parent multicast node  $n_{p(i)}$ . As shown in Figure 8, the first proxy is placed to the node with the highest  $TD$ . Then the value of  $t(i)$  is recomputed for each non-multicast node and the next proxy is placed to the node with the highest  $TD$ . This process continues until  $P$  proxies are placed. Here, the traffic via node  $n_i$  will be recomputed to reflect traffic saving from the newly placed proxies. In addition, delay  $d(i)$  is multiplying to  $t(i)$  to decide which node to place the next proxy. The complexity of this algorithm is  $O(N^2) + O_{\text{routing}}$ .

### 4.3 Greedy algorithm

The greedy algorithm can be used to obtain a near-optimal solution, although its complexity is usually high. Especially when an exhaustive search for the optimal solution is not feasible, it can be used as a measure of how other non-greedy algorithms perform.

**Greedy.** Suppose  $P$  proxies are to be placed. We choose one location at a time. In the first iteration, we presume a proxy to be placed on each of the  $N - 1$  locations and evaluate their objective functions, respectively. The location that yields the best value of the objective function is selected to place the first proxy. In the second iteration, we search for the next best location that, considering both the source and the proxies already placed, yields the smallest value of the objective function. This process continues until all  $P$  proxies are placed. Its complexity is  $O(N^2) \times O_{\text{routing}}$ .

### 4.4 Local search algorithm

Local search was one of the early techniques for combinatorial optimization. It has been applied to solve NP-hard optimization problems [25]. The principle of local search is to refine a given initial solution point in the solution space by searching through the neighborhood of the solution point. We can apply the local search technique to refine the proxy placement solution.

**Local search.** Any of the above algorithms can be used to generate an initial solution. A randomly selected proxy is removed and then placed to a randomly selected non-multicast node. If the initial solution is improved, the proxy remains in the new location. Otherwise, the proxy is restored back to its original location. This process repeats for  $B$  times. The complexity of the local search is  $B \times O(N)$  for SPT tree and  $B \times O_{\text{routing}}$  for other type of trees since the multicast tree needs to be rebuilt for each search. Here,  $B$  is a constant.

**Example 2.** To illustrate various MPP algorithms, another sample graph  $G_{\text{sample2}}$  is shown in Figure 9 with  $N = 15$ , which is generated by the Georgia Tech Internetwork Topology Generator (GT-ITM) [2]. The graph generated by GT-ITM with the transit-stub model closely resembles the Internet hierarchy. Here, we assume that the node weight  $w(i)$  is set to 1. Nodes  $n_0$  and  $n_2$  are transit nodes. Node sets  $(n_4, n_5, n_6, n_7, n_8)$ ,  $(n_9, n_{10}, n_{11})$ ,  $(n_{12}, n_{13}, n_{14})$ ,  $(n_1, n_3)$  are four stubs. Links  $l_{2,14}$  and  $l_{0,1}$  are transit-stub links. Link  $l_{4,9}$  is a stub-stub link. The source is at node  $n_0$ . The numbers listed on links are the  $e(i, j)$ . The exhaustive search of an optimal solution can easily be obtained. The routing algorithm  $A_{\text{DDB}}$  is used in this example, subject to minimizing  $DB_M$ . The results of various MPP algorithms are compared with the optimal solution. Figure 10 shows the proxy placement generated by the Random, Max-degree, Strategic, ASP, Max-traffic, Max-TD, and Greedy algorithms. The optimal solution by exhaustive search is also shown which is the same as the one generated by Greedy. Table 4 shows the delay  $D_M$ , bandwidth consumption  $B_M$ , and delay-bandwidth-consumption  $DB_M$  of these MPP algorithms. The Random placement has the highest  $D_M$  and  $B_M$  values. The Max-Degree placement shows a lower delay and bandwidth consumption, in which non-multicast node  $n_{11}$  is connected to multicast node  $n_9$  instead of node  $n_0$ , resulting in a lower  $DB_M$  value. It is the same in the strategic placement. ASP exhibits the smallest delay but its  $B_M$  value is higher than that of the Max-Traffic, Max-TD and Greedy algorithms. The Max-Traffic and Max-TD algorithms generate the same placement in this small graph. The Greedy algorithm generates the same tree as the optimal placement. Both of them exhibit the smallest  $DB_M$  value, which is the objective function of routing algorithm  $A_{\text{DDB}}$  used here.

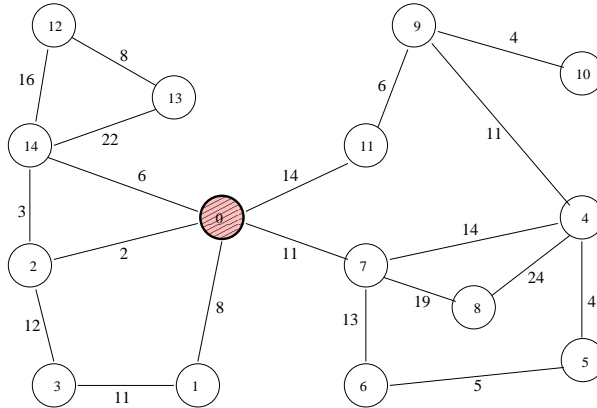
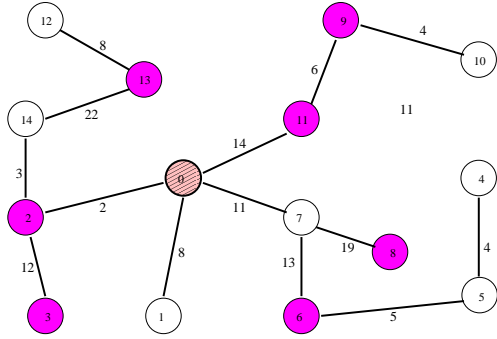
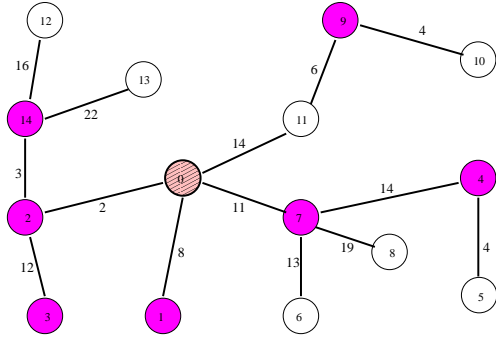


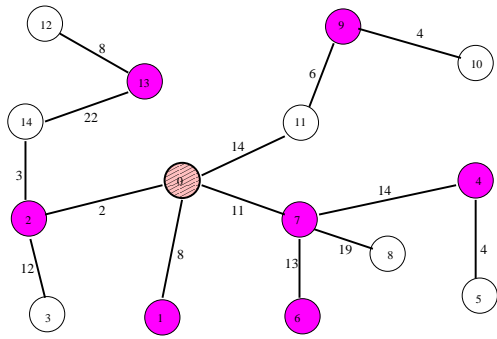
Figure 9: Sample graph  $G_{\text{sample2}}$  for MPP algorithms.



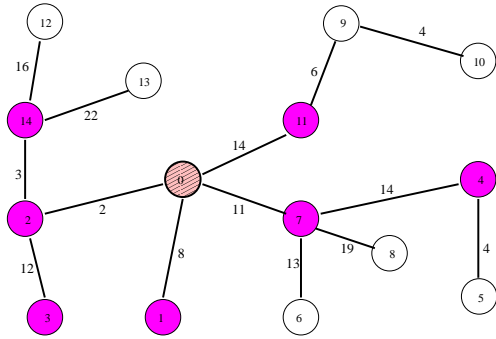
(a) Random



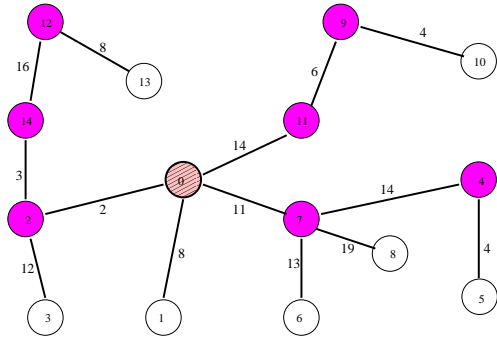
(b) Max-Degree



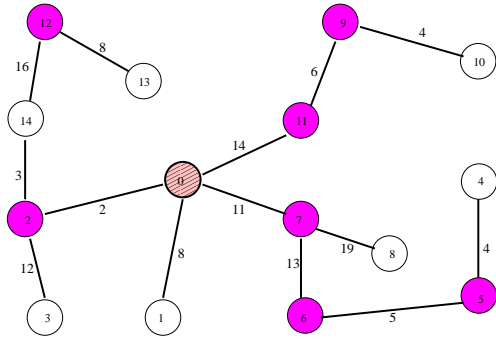
(c) Strategic



(d) ASP



(e) Max-Traffic and Max-TD



(f) Greedy and Optimal

Figure 10: Different placements for  $G_{\text{sample2}}$ .

Table 4:  $D_M$ ,  $B_M$ , and  $DB_M$  of  $G_{\text{sample2}}$  for MPP algorithms

MPP Algorithms	$D_M$	$B_M$	$DB_M$
Random	276	161	44436
Max-Degree	266	154	40964
Strategic	280	149	41720
ASP	254	154	39116
Max-Traffic, Max-TD	256	134	34304
Greedy, Optimal	264	128	33792

## 5 Experimental Results

In this section, placement algorithms with different routing strategies are evaluated. The network graphs are generated by the Georgia Tech Internetwork Topology Generator (GT-ITM) [2]. The transit-stub model is analogous to the current Internet backbone transit networks and local networks. We construct ten different graphs, each consisting of 200 nodes. Each graph is made up of eight transit-domain nodes. The average number of stub domains attached per transit node is three. Each stub domain has eight nodes. The delay of edges is between 1 to 100, with an average of 20. Shown in each figure below, all the results are averaged over the ten graphs. In each instance, a random node is assigned as the source, which remains the same for different placement and routing algorithms. The number of proxies ranges from 1 to 40.

### 5.1 Routing algorithms

We first evaluate multicast trees built by various routing algorithms, in terms of three performance metrics  $D_M$ ,  $B_M$ , and  $DB_M$ . We select two MPP algorithms, Random and Greedy, to generate the placement for this experiment. Four routing algorithms, SPT, MST, P-D, and DDB, are evaluated. Among them,  $M_{\text{SPT}}$  is designed to optimize  $D_M$ ,  $M_{\text{MST}}$  to optimize  $B_M$ , and  $M_{\text{HOT}}$  associated with P-D or DDB is designed to optimize  $DB_M$ .

For comparison, the relative values of the performance are defined here:

$$D = D_M / D_{\text{IPM}}$$

$$B = B_M / B_{\text{IPM}}$$

$$DB = \frac{D_M \times B_M}{D_{\text{IPM}} \times B_{\text{IPM}}} = D \times B$$

where, IPM stands for the traditional IP multicast tree, which is commonly used as a reference point for comparison. Due to the SPT-oriented routing algorithm used in IP multicast,  $D_{\text{IPM}}$  is the minimal value

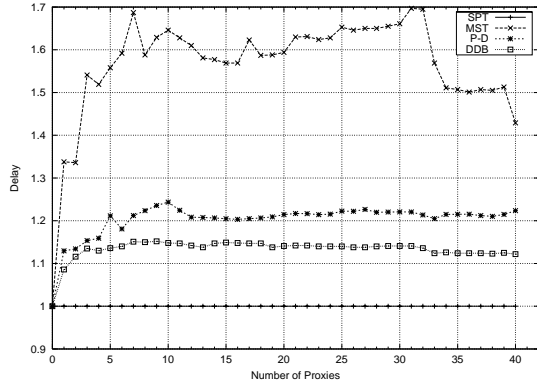
among all possible  $D_M$ . Then, following the same routing, every node acts as an IP multicast router leading to  $B_{IPM}$ .

Figure 11 illustrates D, B, and DB for the given Random and Greedy placements, respectively. It can be seen that graphs in the right column generally show better performance compared to ones in the left column. Evidently, the Greedy placement on the right has produced a better placement than the Random placement on the left. With a good placement, the difference between four routing algorithms, SPT, SMT, P-D, and DDB, is not significant. The difference of the DB value is within 8%. Given a Random placement, good routing algorithms can help to some extent. As shown in Figure 11(e), overall, SPT has minimal delay and highest bandwidth consumption. MST consumes the least bandwidth but has the highest delay. DDB exhibits a much better performance than SPT. The difference of the DB value can be as large as 100%. DDB is the best and its DB value is always the lowest, especially when dealing with the Random placement. However, it is the slowest among the four routing algorithms since it recomputes the metric every iteration. P-D performs slightly worse than DDB; its bandwidth consumption is similar to DDB, but its delay is longer.

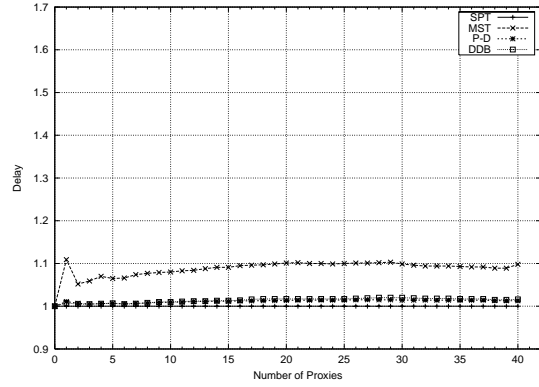
Considering delay only, SPT is obviously superior to the others by its definition. And it is independent as to where proxies are placed. Other routings exhibit increased delay in general, however, the increase in delay should be kept minimal when reducing the bandwidth consumption. An inappropriate placement, such as the Random placement shown in the left column, together with MST routing algorithms can lead to a 34% to 70% increase in the total delay. For details, graphs in Figure 12 show the cumulative distribution of delay D when 20 proxies are placed. The horizontal axis represents a given value of delay D and the vertical axis represents the percentage of individual nodes for which the delay D value was less than this value. DDB and MST behave substantially differently in terms of the total delay. For MST, more than 90% of the nodes have less than 15% delay increase with the Greedy placement, but most nodes have more than 50% delay increase with the Random placement. MST certainly cannot tolerate an inappropriate placement. In contrast, the DDB algorithm is better, where with the Greedy placement, more than 90% of the nodes have less than a 4% delay increase; and with the Random placement more than 90% of the nodes have less than a 25% delay increase.

SPT results in a higher bandwidth consumption, particularly for the Random placement, where it consumes up to 140% more bandwidth than MST does. The bandwidth consumption of DDB is only slightly higher than MST.

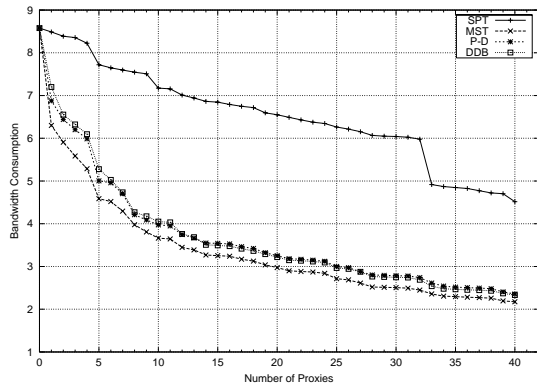
In Figure 13, all routing algorithms are compared with reference to the best case, Greedy placement with DDB routing. Here, the relative performance is defined as the DB value divided by the best DB value across all routing and placement algorithms with the same number of proxies placed. The error-bars in Figure 13 correspond to the minimum, maximum, and average, respectively, of the relative performance of the corresponding routing algorithms. An evident observation is that with a good placement, such as



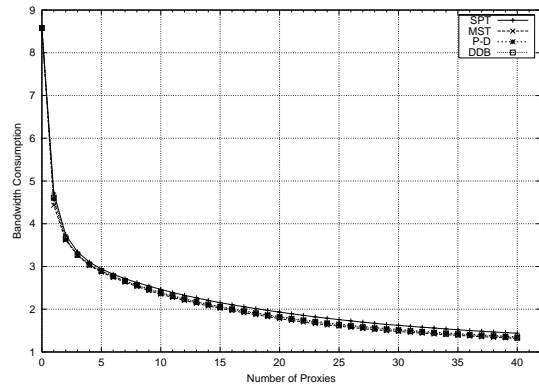
(a) Delay for Random Placement



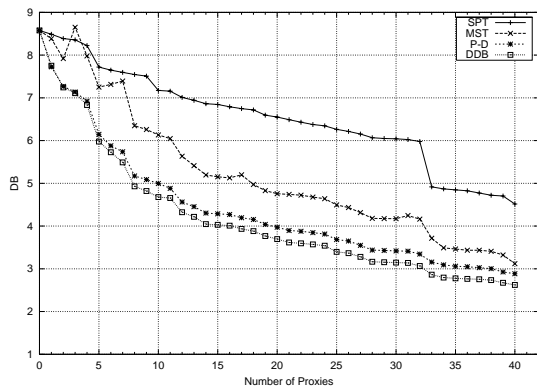
(b) Delay for Greedy Placement



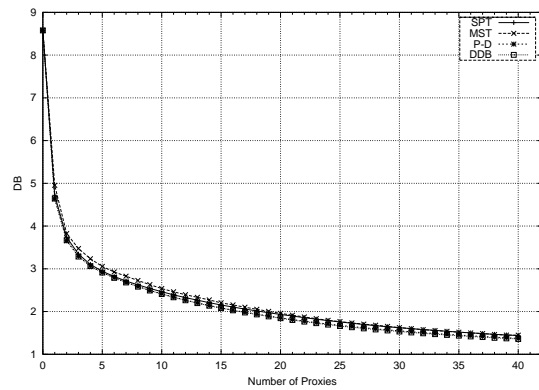
(c) Bandwidth for Random Placement



(d) Bandwidth for Greedy Placement

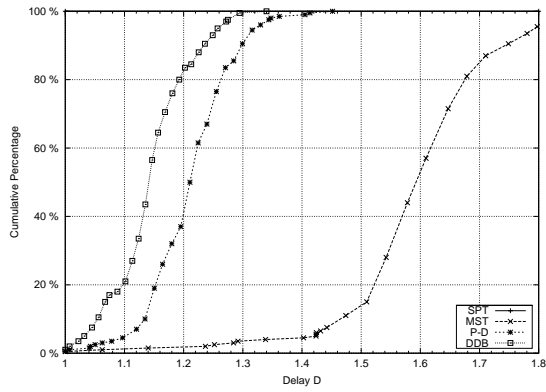


(e) DB for Random Placement

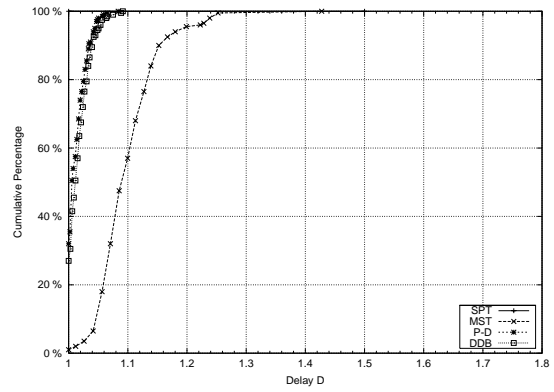


(f) DB for Greedy Placement

Figure 11: D, B, and DB Comparison of Routing Algorithms.

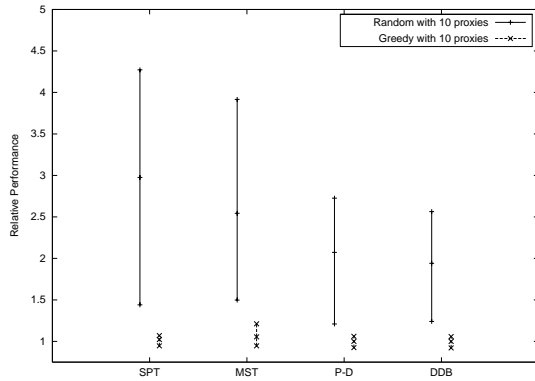


(a) Random Placement

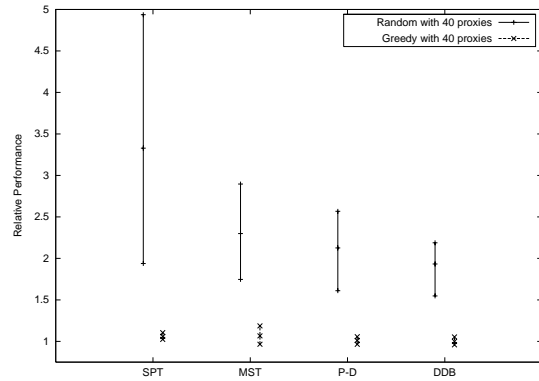


(b) Greedy Placement

Figure 12: Cumulative Delay Percentage from Routing Algorithms.



(a) P = 10



(b) P = 40

Figure 13: Relative Performance of Routing Algorithms.

Greedy, all four routing algorithms perform well. With an inappropriate placement, such as Random, the performance of different routing algorithms is substantially divergent and much worse. Hence, the proxy placement has essential importance.

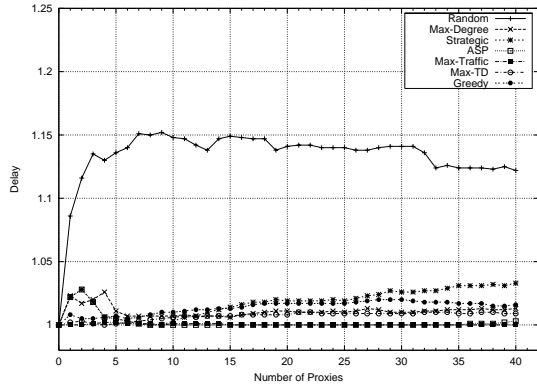
## 5.2 Placement Algorithms

In this section, the multicast trees resulting from various placement algorithms are compared. After the placement is generated, two routing algorithms, SPT and DDB, are used to construct the multicast trees. The resultant D, B, and DB metrics are shown in Figure 14.

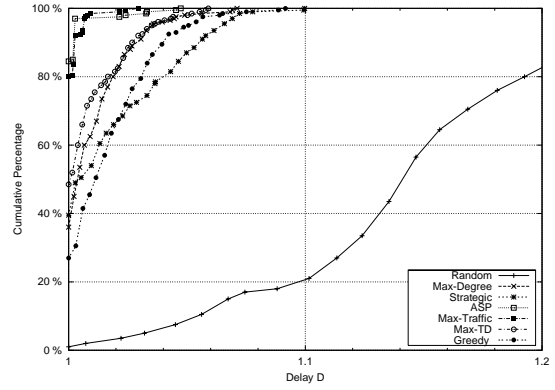
The delay for SPT routing is invariant, so only the delay for DDB is shown in Figure 14(a). The delay of the Random placement is much worse than others. The multicast trees resulting from the Max-Traffic and ASP placement algorithms show the lowest delay. These placement algorithms, except the Random placement, have delay increases of no more than 4% on average. Figure 14(b) shows the cumulative distribution of delay D when 20 proxies are placed. The horizontal axis represents a given value of delay D and the vertical axis represents the percentage of individual nodes for which the delay D value was less than this value. The placement algorithms other than Random have delay increases of no more than 10%.

Figures 14(c) and (d) show the bandwidth consumption. Most placement algorithms significantly reduce the bandwidth consumption when the first five proxies are placed. The bandwidth consumption with five proxies is about two times that of 40 proxies. The Random placement with SPT routing shows large bandwidth consumption. The Max-Traffic placement algorithm requires higher bandwidth consumption when the number of proxies is small, and it outperforms the Max-Degree placement algorithm as more proxies are placed. The Strategic placement and ASP are the same with eight proxies since there are eight transit nodes in the graphs. For other numbers of proxies, ASP outperforms the Strategy placement. The Max-TD placement is exactly the same as Greedy with the SPT routing since the SPT tree is invariant for different numbers of proxies. It is very close to Greedy with the DDB routing, though.

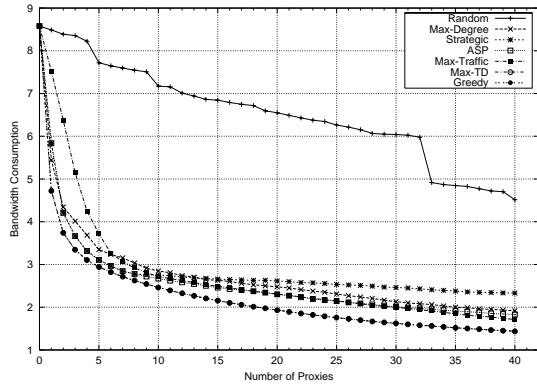
The DB results are illustrated in Figures 14(e) and (f). Since the delay does not significantly change for most placement algorithms, these curves are similar to ones for bandwidth consumption. The Greedy placement produces the best performance, as expected. The Random placement does not produce satisfactory results when SPT routing is used. However, DDB routing can improve the performance of the Random placement, but the DB value of the Random placement is about 100% larger than that of the Greedy placement. Again, the DB value of Max-TD is very close to that of the Greedy algorithm. Since Max-TD is faster, it can serve in place of the Greedy algorithm. In Figure 15, all placement algorithms are compared with reference to the best case, Greedy placement with DDB routing. Here, the relative performance is defined as the DB value divided by the best DB value across all routing and placement algorithms with the same number of proxies placed. Except the Random placement, SPT and DDB perform roughly



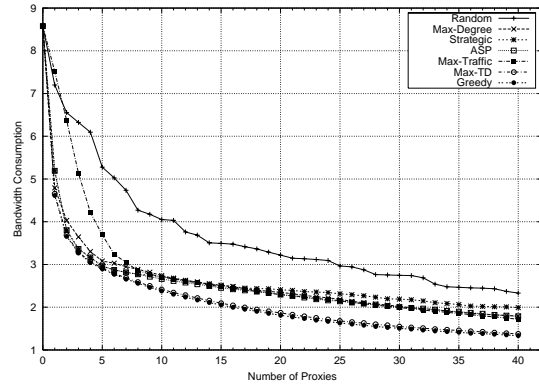
(a) Delay with DDB Routing



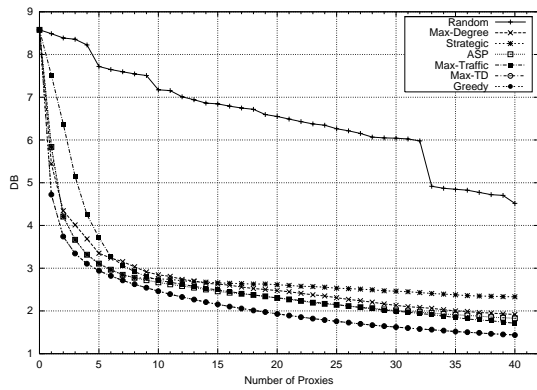
(b) Cumulative Delay Distribution with DDB



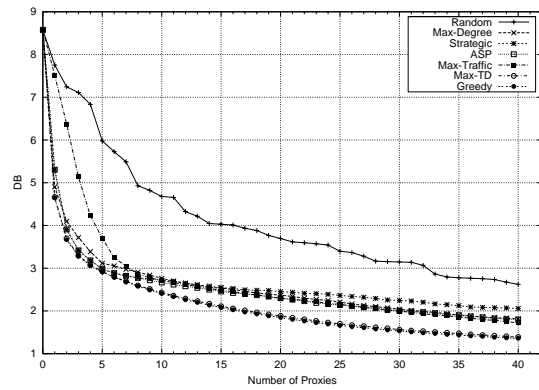
(c) Bandwidth Consumption for SPT Routing



(d) Bandwidth Consumption for DDB Routing

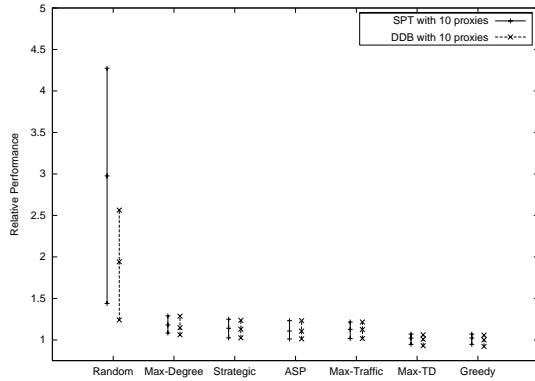


(e) DB for SPT Routing

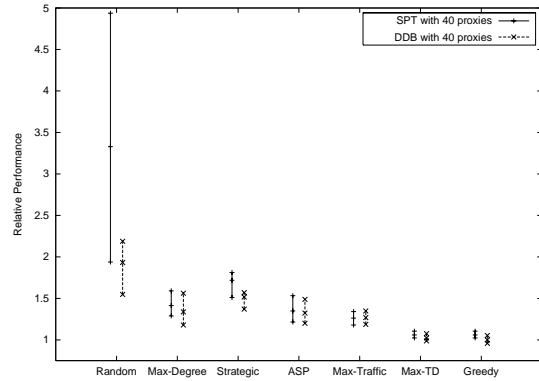


(f) DB for DDB Routing

Figure 14: D, B, DB Comparison of Placement Algorithms.



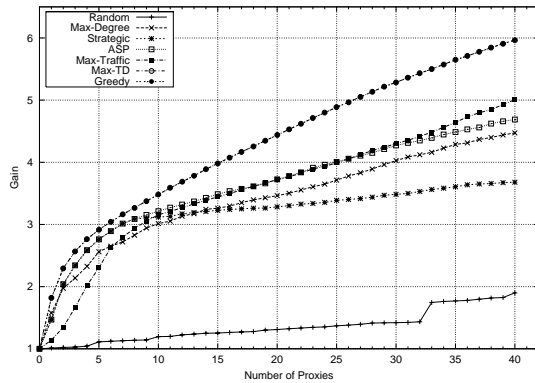
(a)  $P = 10$



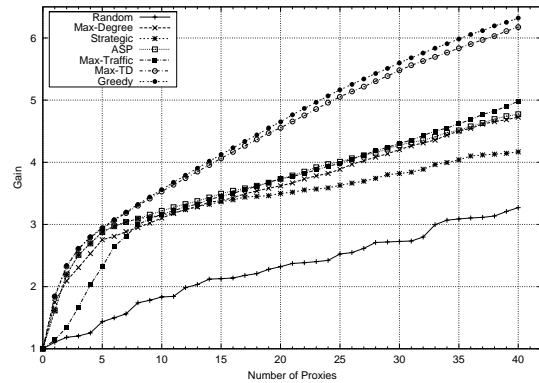
(b)  $P = 40$

Figure 15: Relative Performance of Placement Algorithms.

the same, once again concluding that the placement is important.



(a) Gain with SPT Routing



(b) Gain with DDB Routing

Figure 16: Gain with Proxy Placement.

The benefit of placing proxies can be measured by the *gain*, which is defined as the DB value with no proxy placed divided by the DB value with  $P$  proxies placed. Figure 16 illustrates the gain of various placement algorithms with SPT and DDB routing. The gain of Random is very low, especially for the SPT routing. The popular Strategic placement performs well for a few proxies, but not as good as other placement algorithms for more proxies. Greedy and Max-TD produce the highest gains.

## Local Search

The Greedy placement algorithm produces satisfactory results but with high complexity. Max-TD can produce good performance with moderate complexity. Max-Degree, Max-Traffic, Strategic and ASP are fast, but their performance is not as good as Max-TD and Greedy. Random placement is not satisfactory. However, with low-complexity local search, their performance can be significantly improved. Figure 17 shows the improvement of Random, Max-Degree, and Max-Traffic with local search. An improvement of 12% to 37% can be obtained in 300 iterations. The local search algorithm has been applied to Greedy and Max-TD. Almost no improvement has been observed, possibly due to the fact that their placement is quite close to optimal.

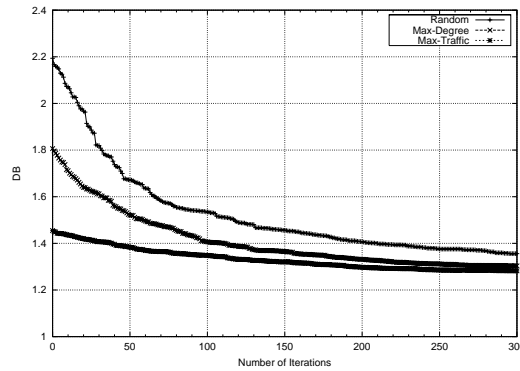


Figure 17: Local search for Random, Max-Degree, and Max-Traffic.

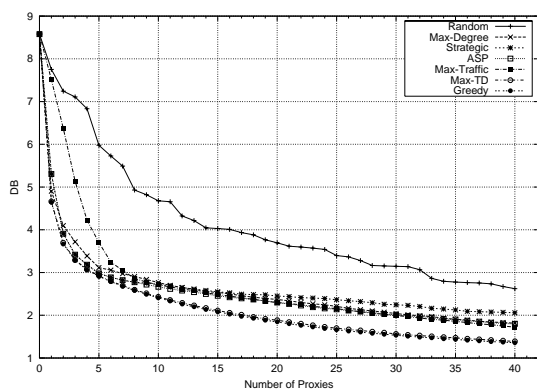
## Low-connectivity and high-connectivity

In order to study how connectivity of graphs impacts the performance of routing and placement algorithms, ten additional graphs were generated with high-connectivity compared to the regular ten graphs with low-connectivity. There are more links in transit and stub domains, between transit and stub domains, and among stub domains in this group of high-connectivity graphs. The graph generation is controlled by four parameters, that is, the number of transit-stub edges, the number of stub-stub edges, the  $\alpha$  value of transit domain, and the  $\alpha$  value of stub domain, which are shown in Table 5. The number of transit-stub edges is the number of extra links connected between transit nodes and stub nodes. The number of stub-stub edges is the number of extra links connected between different stub domains. The  $\alpha$  value, between 0 and 1, controls the connectivity in a transit or stub domain.

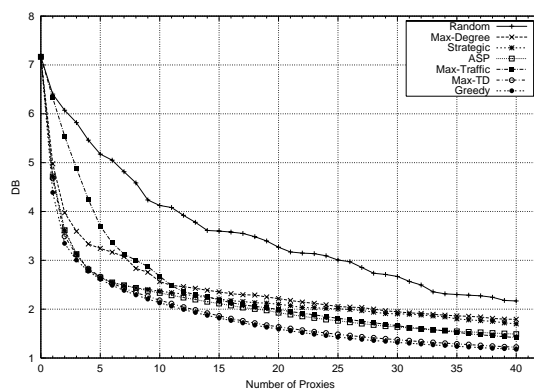
Figure 18 compares the low-connectivity and high-connectivity graphs. The low-connectivity graphs have higher delay and higher bandwidth consumption. This is because in a high-connectivity graph, more paths from the source to destination nodes are available and shorter paths can be selected. In addition, various placement algorithms on high-connectivity graphs diverge from each other in terms of their DB values compared to the other case. Since alternative paths provide more chances for better routing, a good

Table 5: Parameters for graph generation

	Low-connectivity	High-connectivity
# of transit-stub edges	0	5
# of stub-stub edges	0	10
$\alpha$ value of transit domain	0.2	0.6
$\alpha$ value of stub domain	0.2	0.4



(a) Low-Connectivity Graphs



(b) High-Connectivity Graphs

Figure 18: Comparison of Two Groups of Graphs.

algorithm is able to select the best places for proxies.

### Repeated traffic

Figure 19 shows the repeated traffic. Reduction of repeated traffic is important for “hot spot” elimination. With five proxies, the repeated traffic is reduced to 40% for Greedy placement with DDB routing, and with 40 proxies, it is reduced to 7%. There is no repeated traffic when 110 proxies are placed, which is not shown in the figure.

In summary, proxy placement is crucial to the system performance. Random placement is not acceptable, though a good routing algorithm can help. The Greedy algorithm is the best, and Max-TD is very close to Greedy. DDB routing balances well the requirements of low delay and low bandwidth consumption. With a good proxy placement, the increase of delay and bandwidth consumption by DDB is within a few percents compared to that generated by SPT and MST, respectively. A combination of Max-TD placement and DDB routing is able to produce satisfactory performance. Compared to the popular combination of Strategic placement and SPT routing, a 70% gain in performance can be achieved.

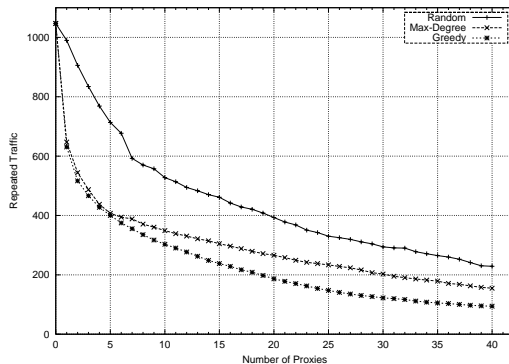


Figure 19: Repeated Traffic with DDB Routing.

## 6 Related Work

Server-based multicast can solve many problems. Compared to the IP multicast, server-based multicast can handle heterogeneous networks and clients. Server-based multicast is able to adapt to different bandwidth and solves the congestion control problem. Retransmission can be handled locally for reliable multicast. Server-based multicast can also work across both space and time by using proxy servers as caches [31, 32, 29]. Existing works on the server-based multicast use strategic placement and various routing algorithms to build multicast trees [5, 6, 14, 22]. A comparison study showed that these routing algorithms normally involve a 200% to 300% longer delay than IP multicast [33]. Many of them are unable to reduce the bandwidth consumption to minimal. Their bandwidth consumption is several times larger than that given by the Minimal Spanning Tree and in certain situation even larger than that of unicast. This paper has investigated both routing and placement problems in a systematic approach. Various algorithms and their system integration are discussed and evaluated to illustrate their performance impacts. These results can be valuable references for other server-based multicast overlay network. Moreover, with power of localization, the application of near-optimal algorithms on both routing and placement becomes feasible [4, 19].

Formulation and evaluation of the Multicast Proxy Placement (MPP) problem is a focal point of this paper, which to our knowledge is the first work formulating the MPP problem. recently, another work [24] considered the server placement problem in overlay networks. However, the objective function in the work is to minimize the number of proxies that cover the entire network with a bounded delay from clients to a proxy. The delay from the source to proxies is unbounded, therefore, there could be a long delay between the source and clients. Furthermore, the bandwidth requirement was not considered in the work.

Work in [23, 21, 20, 16, 15] studied the problem of placing cache replicas or proxies in the network. [23] formulated it as the two well-studied problems: the facility location problem [7] and the  $k$ -median

problem [3]. Now, we use the same graph  $G_{\text{sample1}}$  in Figure 1 to show the difference between cache proxy placement and multicast proxy placement. The cache placement problem can be modeled as the minimum  $k$ -median problem [23]. Its objective function is the system cost

$$C = \sum_{i=1}^N w(n_i) \times d(n_i, c(n_i, P)),$$

where  $w(n_i)$  is the node weight representing the traffic generated by node  $n_i$ ,  $d(n_i, n_j)$  is the length of the shortest path from  $n_i$  to  $n_j$ , and  $c(n_i, P)$  is the proxy in the proxy set  $P$  that is closest to  $n_i$ . Figure 20(a)

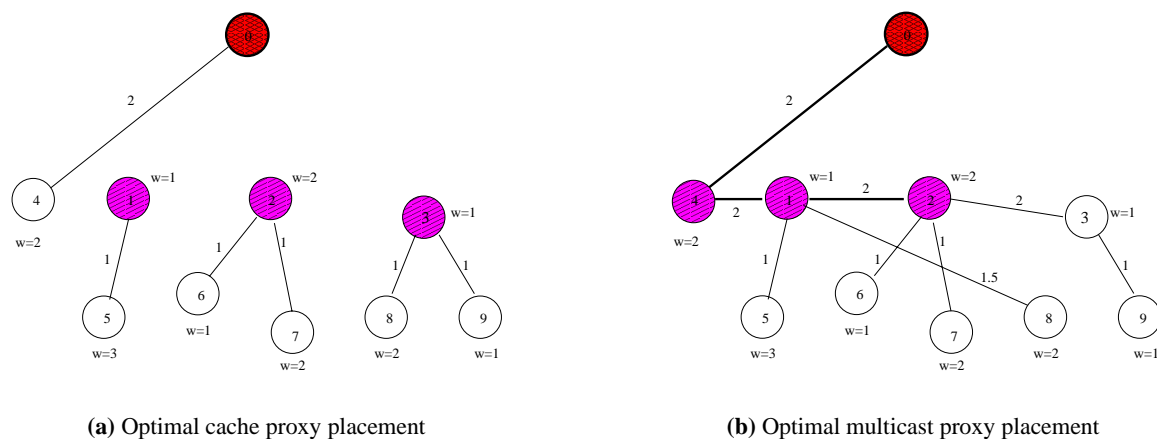


Figure 20: Placement of Cache Proxy and Multicast Proxy.

shows optimal cache proxy placement for three proxies. The proxies are placed to nodes  $n_1, n_2$ , and  $n_3$  and the system cost  $C = 13$ . On the other hand, the optimal multicast proxy placement that generates smallest bandwidth consumption is shown in Figure 20(b). The proxies are placed to nodes  $n_1, n_2$ , and  $n_4$  and the total bandwidth consumption  $B = 20$ . Note that,  $C = 14$  when proxies are placed to nodes  $n_1, n_2$ , and  $n_4$ . On the other hand,  $B = 21$  when proxies are placed to nodes  $n_1, n_2$ , and  $n_3$  as shown in Figure 2(b). The MPP problem has a different objective function from that of the cache proxy placement problem. First, a multicast tree must be built among the multicast nodes. Second, the non-multicast nodes are not necessarily connected to the nearest multicast node except that for the MST tree. It cannot be modeled by the minimum  $k$ -median problem.

## 7 Conclusion

This paper addressed the multicast proxy placement problem. The problem has been formulated and objectives are presented. Based on the performance metrics, a number of routing and placement algorithms have been presented. The performance study showed that the proxy placement is more crucial than routing

in terms of their impact on performance. The Greedy algorithm produces the best solution among these placement algorithms and Max-TD is close to optimal.

An open problem is distributed decision on the proxy placement. Distributed versions of presented algorithms, which we are currently working on, may produce near-optimal results. Yet another direction is to adapt the localization approach. With the power of localization, optimal decision within a zone under single administration may result in global optimization in a federation of zones.

## References

- [1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley, 1974.
- [2] GT-ITM at Georgia Tech University. GT-ITM: Modeling topology of large internetworks. <http://www.cc.gatech.edu/projects/gtitm/>.
- [3] Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *IEEE 40th Annual Symposium on Foundations of Computer Science*, October 1999.
- [4] Y. Chawathe and M. Seshadri. Broadcast federation: An application-layer broadcast internetwork. In *the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, May 2002.
- [5] Yatin Chawathe. *Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service*. PhD thesis, Department of EECS, UC Berkeley, December 2000.
- [6] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *ACM Sigmetrics*, 2000.
- [7] F.A. Chudak and D. Shmoys. Improved approximation algorithms for capacitated facility location problems. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999.
- [8] S. Deering. *Multicast routing in a datagram internetwork*. PhD thesis, Stanford University, 1991.
- [9] S. Deering and David R. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8(2):85–111, May 1990.
- [10] M. Doar and I. Leslie. How bad is naive multicast routing? In *INFOCOM*, April 1993.
- [11] Paul Francis. Yoid: Your own internet distribution. Technical Report at [www.aciri.org/yoid](http://www.aciri.org/yoid), UC Berkeley ACIRI Tech Report, April 2000.
- [12] David Helder and Sugih Jamin. Banana tree protocol, an end-host multicast protocol. Technical Report CSE-TR-429-00, Univ. of Michigan, 2000.
- [13] H. Holbrook and D. Cheriton. IP multicast channels: EXPRESS support for large-scale single source applications. In *ACM SIGCOMM*, September 1999.
- [14] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O’Toole Jr. Overcast: Reliable multicasting with an overlay network. In *5th Symposium on Operating System Design and Implementation (OSDI)*, December 2000.
- [15] X. Jia, D. Li, X. Hu, and D.Z. Du. Placement of read-write web proxies on the internet. In *21st International Conference on Distributed Computing Systems*, 2001.

- [16] K.M. Kamath, H.S. Bassali, R.B. Hosamani, and L.Gao. Policy-aware algorithms for proxy placement in the internet. In *ITCOM*, 2001.
- [17] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problem II: the p-medians. *SIAM Journal Application Math.*, 37:539–560, 1979.
- [18] A. Kershenbaum, P. Kermani, and G. Grover. Mentor: An algorithm for mesh network topological approximation and routing. *IEEE Trans. on Communications*, 39:503–513, 1991.
- [19] Christopher Kommareddy. Scalable application layer multicast. In *ACM SIGCOMM*, August 2002.
- [20] P. Krishnan, D. Raz, and Y. Shavitt. The cache location problem. *IEEE/ACM Transaction on Networking*, 8:568–582, October 2000.
- [21] B. Li, M.J. Golin, G.F. Italiano, X. Deng, and K. Sohrawy. On the optimal placement of web proxies in the internet. In *IEEE Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, 1999.
- [22] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *3rd Usenix Symposium on Internet Technologies and Systems (USITS)*, March 2001.
- [23] Lili Qiu, Venkata N. Padmanabhan, and Geoffrey M. Voelker. On the placement of web server replicas. In *INFOCOM*, April 2001.
- [24] S. Shi and J. Turner. Placing servers in overlay networks. In *Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, July 2002.
- [25] R. Sosič and J. Gu. Local search for the satisfiability (SAT) problem. *IEEE Trans. on Systems, Man, and Cybernetics*, 23(3):1108–1129, July 1993.
- [26] Ion Stoica, T. S. Eugene Ng, and Hui Zhang. Reunite: A recursive unicast approach to multicast. In *INFOCOM*, 2000.
- [27] Liming Wei and Deborah Estrin. A comparison of multicast trees and algorithms. In *INFOCOM*, April 1994.
- [28] Liming Wei and Deborah Estrin. The trade-offs of multicast tree and algorithms. In *International Conference on Computer Communications and Networks*, August 1994.
- [29] M.Y. Wu, S. Ma, and W. Shu. Scheduled video delivery for scalable on-demand service. In *International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSS-DAV)*, May 2002.
- [30] M.Y. Wu and W. Shu. A cache network to eliminate hot spots for live and near-live content distribution and delivery, 2002. submitted.
- [31] M.Y. Wu and W. Shu. Edge station in access networks for video distribution. In *International Conference on Computers and Their Applications (CATA-2002)*, April 2002.
- [32] M.Y. Wu and W. Shu. Efficient support for interactive browsing operations in clustered CBR video servers. *IEEE Trans. on Multimedia*, 4, March 2002.
- [33] Yan Zhu. Comparative study of server-based multicast overlay network. Master’s thesis, Department of Electrical and Computer Engineering, University of New Mexico, 2002.