Telecommunication Systems 16:3,4, 483–501, 2001
© 2001 Kluwer Academic Publishers. Manufactured in The Netherlands.

Aggregating Multicast Demands on Virtual Path Trees

MICHAEL MONTGOMERY [*] and GUSTAVO DE VECIANA [**]
Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712-1084, USA

Abstract. In ATM networks, Virtual Path Connections (VPCs) are not only a device for bundling Virtual Circuit Connections (VCCs), but they serve as an intermediate resource management layer wherein trade-offs between cost, complexity, and efficiency are likely to be made on a slower time scale than connection holding times. In this paper we consider bundling multicast connections on VP trees. In particular, we show that the aggregation of multicast demands onto large VP trees can be an effective way to reduce capacity requirements, balance network loads, and reduce the number of VP trees required.

Keywords: aggregation, virtual paths, multicast, ATM

1. Introduction

In ATM networks, Virtual Paths (VPs) that can carry multiple Virtual Circuits (VCs) are allocated to reduce the complexity of call setup and traffic management at the possible expense of efficiency. A VP layout consists of a vector of capacities allocated to VPs that are set up on a subset of network routes on a permanent or semi-permanent basis. This logical partitioning may be done periodically for the purpose of adaptive resource allocation due to changing network conditions.

Given a set of multicast demands for a network incorporating multicast-capable switches, a layout of VP trees could be established using an algorithm such as the one found in [7]. Setting up a Switched Virtual Circuit (SVC) tree requires significant signaling resources, so even with low multicast demands, creating and using VP trees on a slower time scale than connection holding times can be worthwhile. The decision to use SVC or VP trees will ultimately be determined by which resource is the bottleneck: call processing capacity or bandwidth, as well as possible setup delay constraints and the future demand for multicast connections. If call processing capacity is limited, the use of VP trees will be necessary. If bandwidth is scarce, SVC trees will be needed to use the bandwidth in the most efficient manner possible.

The graph shown in figure 1 exhibits the call processing load in calls per second (cps) versus the demand, measured in Erlangs, and the mean holding times of connec-

[*] M. Montgomery is now with the Center for Information Infrastructure Technology at the Y-12 National Security Complex, 1099 Commerce Park, Oak Ridge, TN 37830, USA. E-mail: montgomerymc@y12.doe.gov.

[**] G. de Veciana is supported by National Science Foundation Career Grant NCR-9624230 and by Southwestern Bell Technology Resources Inc. E-mail: gustavo@ece.utexas.edu.

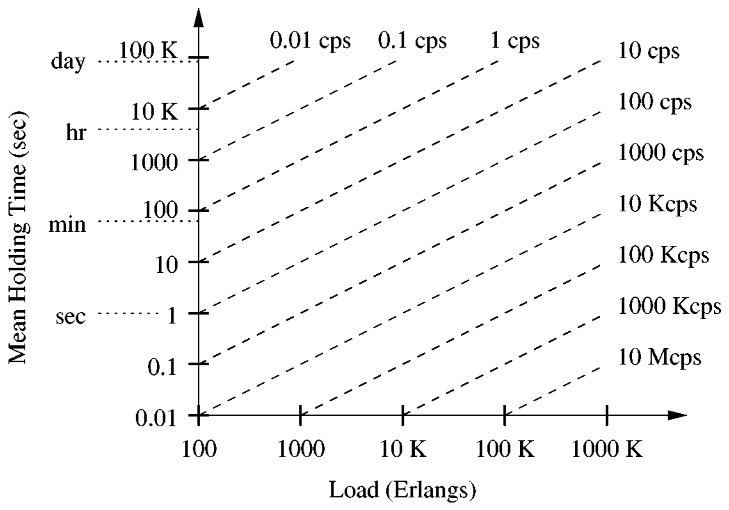

Figure 1. Call processing rates versus demand and mean holding times.

tions.[1] As a reference point, the current capabilities of ATM core switches range from 385 cps for the FORE Systems ForeRunner ASX-200BX to 5000 cps for the Ascend GX 550 where the call processors are implemented in hardware with a separate processor card per port. Note however that a switch might need to support full accounting capabilities, such as authentication and the generation of billing records on a per connection basis, therefore it is difficult to extrapolate what throughput might be achieved. Clearly, if high demand develops for multicast connections with short holding times, on the order of seconds or minutes, then VP trees would be beneficial in limiting the amount of signaling and call processing performed in the network. In addition, even if the average demand can be sustained, call requests arrive as stochastic processes, and the variability in the rate of requests is likely to lead to periods of focused congestion with eventual lost calls due to a lack of signaling resources, further strengthening the case for VP trees.

Another possible concern is that one recent empirical study [10] has shown that setup delays for SVCs can be significant, especially for multicast, although this should improve as ATM signaling software matures. In contrast, if an appropriate VP tree is already established, the setup delay would be minimal since call processing is only necessary at the VPC end nodes.

Given that we want to establish and use VP trees at least some of the time, in this paper we argue that due to statistical multiplexing, one may actually save capacity by aggregating heterogenous multicast demands on the same VP tree. Taking advantage of this fact, we propose a pre- or post-processing step to the VP multicast layout problem, which either reduces the complexity of the required optimization or further improves

[1] Note that the demand, measured in Erlangs, i.e., average connection requests per mean connection holding time, is also a rough estimate for the average number of concurrent connections in the system.

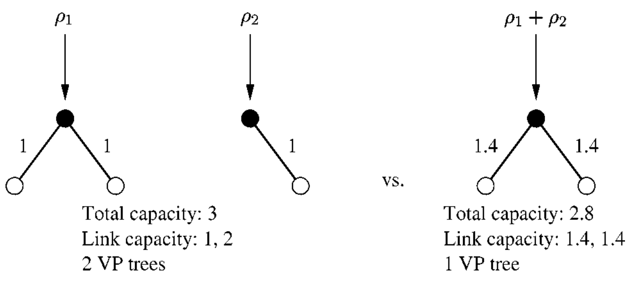

Figure 2. In this example, aggregating the multicast demands provides capacity savings, better load balancing, and a reduction in VP setup and management loads.

upon obtained solutions. If the VP multicast layout is already determined, then our procedure could be a post-processing step; if not, it would be a pre-processing step which is discussed further in section 4. The need for aggregating multicast demands onto VP or VC trees to avoid VP/VC explosion has previously been suggested in the context of an IP over ATM environment [3]. Suppose we have a destination set of size 10, where a "destination set" is a candidate group of destinations for which any subset may be the recipient of a multicast connection from a given source. Then there are 1023 possible non-empty subsets of destinations, and it is likely to be impractical to set up a separate tree for each subset with nonzero demand.

Consider the situation illustrated in figure 2. There is a single source and two destinations with a demand ρ_1 for multicast connections to both destinations and a demand ρ_2 for unicast connections to only one destination. Suppose that a capacity of 1 on each link is able to accommodate the demands at the desired call blocking probability. Furthermore, suppose that if the demands are aggregated, a capacity of 1.4 on each link is required. In this case, despite the fact that connections of type 2 are needlessly using both links, we obtain benefits from aggregating the multicast demands in three areas: the total required capacity is less, the link capacities are more evenly balanced, and one VP tree, rather than two, is required.

More generally, suppose we are given demands ρ_1 and ρ_2 for multicast connections requiring unit bandwidth from a common source to destination sets $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively, where $\mathcal{D}_2 \subseteq \mathcal{D}_1$.[2] Assuming the network switches have the proper multicast capabilities, we want to establish VP trees for sets $\mathcal{D}_1$ and $\mathcal{D}_2$. The question is whether we should establish two separate trees or a single tree to accommodate the demands ρ_1 and ρ_2, i.e., will the benefit of additional multiplexing at the call and cell levels outweigh the bandwidth wasted by connections for the smaller set $\mathcal{D}_2$ using the larger tree? This situation is illustrated in figure 3.

To answer this question, we first introduce the function $\alpha(\rho, B)$ which gives the link capacity needed to accommodate the demand ρ at a specified call blocking prob-

[2] Of course, one destination set does not have to be a subset of another, but this case leads to the simplest algorithms and the greatest potential savings. In the more general case, a shared VP tree would have to reach destinations in the smallest set containing both $\mathcal{D}_1$ and $\mathcal{D}_2$.

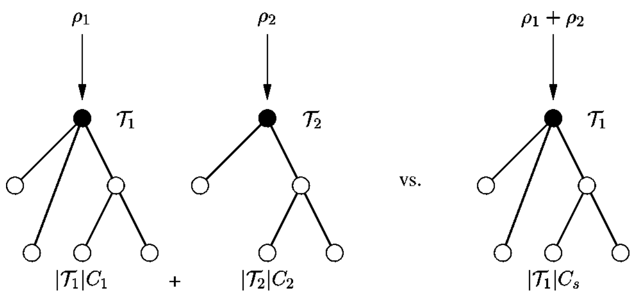

Figure 3. Establishing separate VP trees to accommodate demands ρ_1 and ρ_2 would require a total capacity of $|\mathcal{T}_1|C_1 + |\mathcal{T}_2|C_2$. Sharing the larger tree would require capacity $|\mathcal{T}_1|C_s$.

ability B. This function may account for statistical multiplexing at the call level, the burst level, the cell level, or some combination of the three. To determine the capacity needed for the VPs in each case, we solve for $C_1 = \alpha(\rho_1, B)$, $C_2 = \alpha(\rho_2, B)$, and $C_s = \alpha(\rho_1 + \rho_2, B)$. For separate VP trees $\mathcal{T}_1$ and $\mathcal{T}_2$, the total capacity needed is $C = |\mathcal{T}_1|C_1 + |\mathcal{T}_2|C_2$, where $|\mathcal{T}|$ is the number of links in multicast tree $\mathcal{T}$. When sharing tree $\mathcal{T}_1$, the total capacity needed is $C' = |\mathcal{T}_1|C_s$. If $C' < C$, it would be beneficial to use a single tree. We can rewrite this condition as

$$\frac{C_s - C_1}{C_2} < \frac{|\mathcal{T}_2|}{|\mathcal{T}_1|}. \tag{1}$$

It should be noted that there are additional benefits to sharing VP trees besides capacity savings: e.g.,

- savings in Virtual Path Identifier (VPI) usage – sharing VP trees would reduce the number of VPs, and hence VPIs, needed for a particular layout,
- a reduction in VP setup and management loads as well as setup delays for a connection,
- a more even balancing of load across the network – sharing VP trees tends to reduce the variance in the allocated link capacities, and
- possible savings in the size of buffers needed at the input of each VP tree due to the increased cell level multiplexing from combining demands.

In section 2, we explore the use of the Erlang B formula to implicitly determine the function α, and then further consider a Gaussian traffic model. Given an initial collection of destination sets, heuristics for finding an aggregation of demands requiring the least total capacity are proposed and evaluated through simulation in section 3. Finally, in section 4, we present methods for dealing with unknown topologies, and section 5 concludes the paper with a few additional comments.

2. Specific examples

Herein, we assume a large population (infinite sources) model where the requests for multicast connections arrive as Poisson processes.[3] In this case, the Erlang B formula can be used to implicitly determine the function α and assess the benefits of call level multiplexing alone [14]. For the above setup, we must solve $E(\rho_1, C_1) = E(\rho_2, C_2) = E(\rho_1 + \rho_2, C_s) = B$ for C_1, C_2, and C_s, and then test the condition for sharing the larger tree as expressed in (1). Based on the Erlang function and a blocking probability of 10^{-3}, figure 4 shows the threshold for capacity savings (maximum value of ρ_2 for a given ρ_1) as $|\mathcal{T}_2|/|\mathcal{T}_1|$ is varied from 0.5 to 0.9. Below each line, the capacity savings is greater than zero. Note that, because of the sub-additivity property explained below, the threshold for $|\mathcal{T}_2|/|\mathcal{T}_1| = 1$ would be a vertical line at $\rho_1 = 0$.

For a constant blocking probability B, the function $C = g(\rho)$, defined implicitly by Erlang's formula, is concave and sub-additive [14].[4] The sub-additivity property, i.e., $g(\rho_1 + \rho_2) < g(\rho_1) + g(\rho_2)$, implies that two separate links require more capacity than a single link with the same total traffic, or in other words, it assesses the benefits of call level multiplexing. Furthermore, as the desired call blocking probability is decreased, the multiplexing benefits get better. However, even for a modest blocking probability of 10^{-3}, we can achieve significant capacity savings from aggregation.

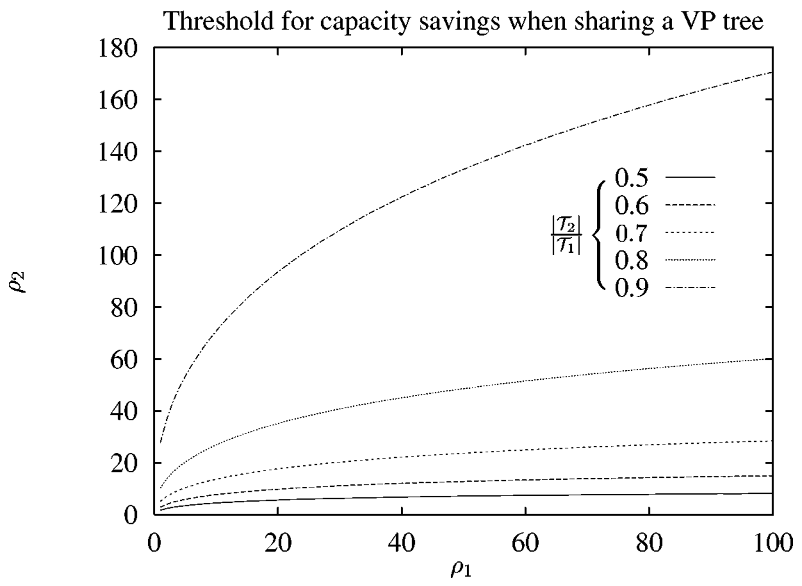

Figure 4. For a blocking probability of 10^{-3}, the threshold for capacity savings is plotted for values of $|\mathcal{T}_2|/|\mathcal{T}_1|$ ranging from 0.5 to 0.9. Below each line, the capacity savings is greater than zero.

[3] Note that aggregating demands and sharing a tree will increase the population of sources requesting access to the tree and will serve to strengthen this assumption.

[4] For a plot of the inverse Erlang function $C = g(\rho)$ for various values of B, see [9].

The potential capacity savings are graphed for some different scenarios in figures 5–9 for a call level blocking probability of 10^{-3}. Figures 5–7 exhibit the potential savings in capacity when aggregating 2 multicast groups. The savings are given by $100(C - C')/C$, where C and C' are the total required capacities for separate VP trees and a shared VP tree, respectively, so the values shown are relative savings percentages. Figure 5 shows results for the situation shown in figure 3 where $|\mathcal{T}_1| = 5$ and $|\mathcal{T}_2| = 4$. The tree sizes are varied in the remaining figures. In figures 8 and 9, there are m multicast groups with $\mathcal{D}_1$ being the largest destination set, and the savings graphed in the figures are a comparison between using m separate VP trees and aggregating all m groups onto a single tree $\mathcal{T}_1$.

Figures 5–7, and 9 show that low values of $\rho_1, \rho_2, \ldots, \rho_m$ lead to higher savings. This is due to the fact that the savings percentage from the sub-additivity of the inverse Erlang function, $C = g(\rho)$, introduced above is greater for smaller values of ρ. Note that the low offered load regime is not unrealistic in practice. Indeed, currently the utilizations achieved on ATM/SONET links are as low as 10–20% due to overheads and spare capacity reserved for $1 + 1$ failure protection [12]. This would mean that a 10 Gbps link would have around 2 Gbps of usable capacity. When this capacity is further partitioned into a logical VP layout, with say 100 VPs, then each VP would have a capacity of 20 Mbps. Each VP could, for instance, accommodate 20 video connections at 1 Mbps, or in other words, the VP capacity would be $C = 20$ circuits.

Aggregation of larger trees also leads to higher savings, as can be seen in figures 6–9, because the wasted bandwidth due to a call for a smaller set of destinations using the

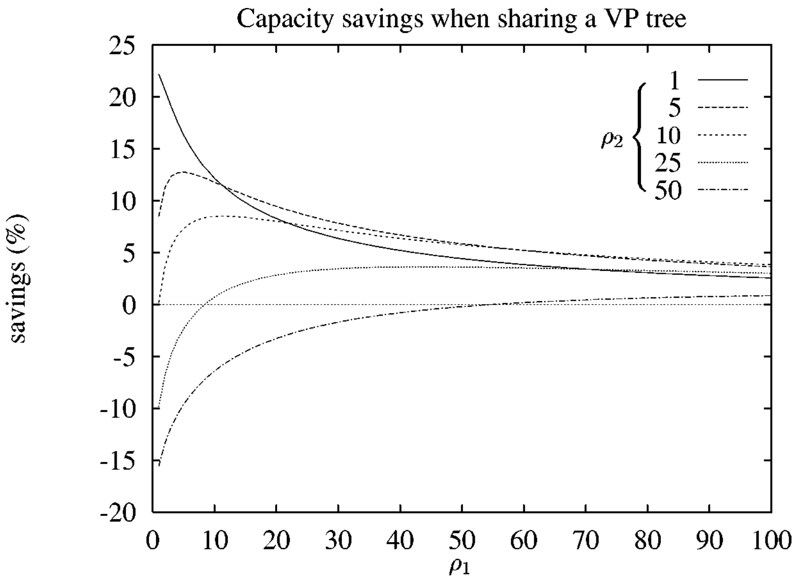

Figure 5. The capacity savings shown are for sharing VP tree $\mathcal{T}_1$ between 2 multicast groups with offered loads ρ_1 and ρ_2, fixed tree sizes $|\mathcal{T}_1| = 5$ and $|\mathcal{T}_2| = 4$, and a blocking probability $B = 0.001$.

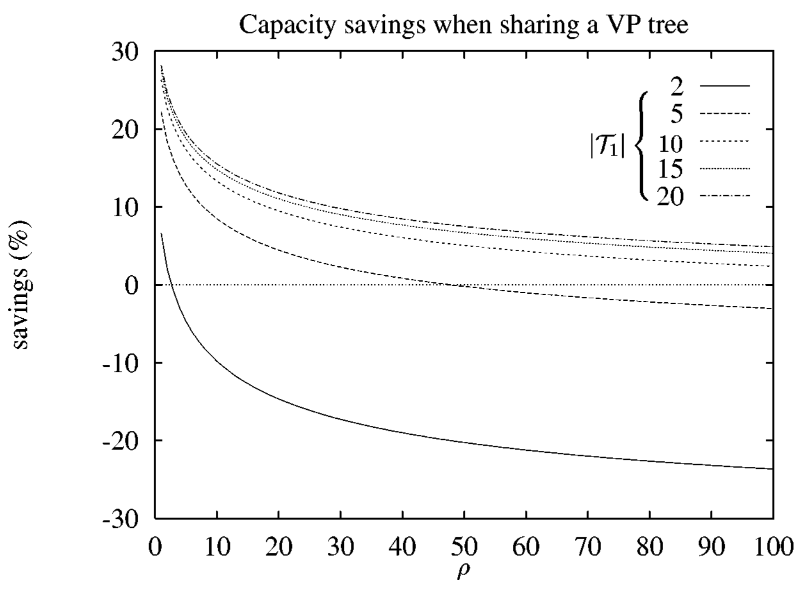

Figure 6. The capacity savings shown are for sharing VP tree $\mathcal{T}_1$ between 2 multicast groups with offered loads $\rho_1 = \rho_2 = \rho$, a smaller tree size $|\mathcal{T}_2| = |\mathcal{T}_1| - 1$, and a blocking probability $B = 0.001$.

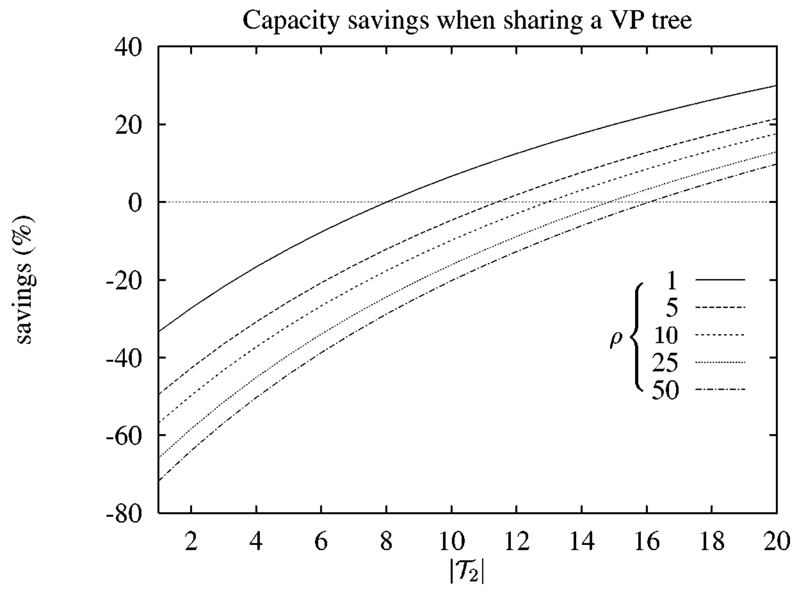

Figure 7. The capacity savings shown are for sharing VP tree $\mathcal{T}_1$ between 2 multicast groups with offered loads $\rho_1 = \rho_2 = \rho$, a larger tree size $|\mathcal{T}_1| = 20$, and a blocking probability $B = 0.001$.

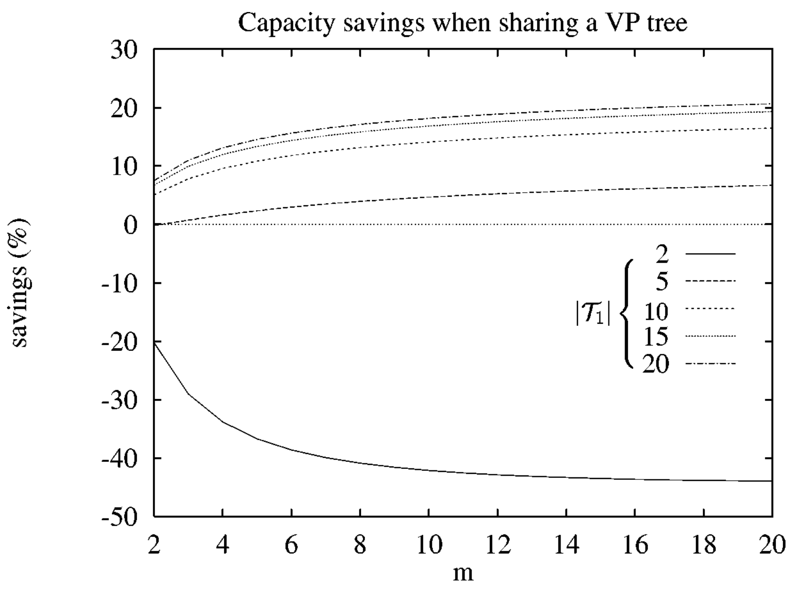

Figure 8. The capacity savings shown are for sharing VP tree $\mathcal{T}_1$ between m multicast groups with destination sets $\mathcal{D}_i \subseteq \mathcal{D}_1$ and tree sizes $|\mathcal{T}_i| = |\mathcal{T}_1| - 1$ for groups $i = 2, 3, \ldots, m$, offered loads $\rho_1 = \rho_2 = \cdots = \rho_m = 50$, and a blocking probability $B = 0.001$.

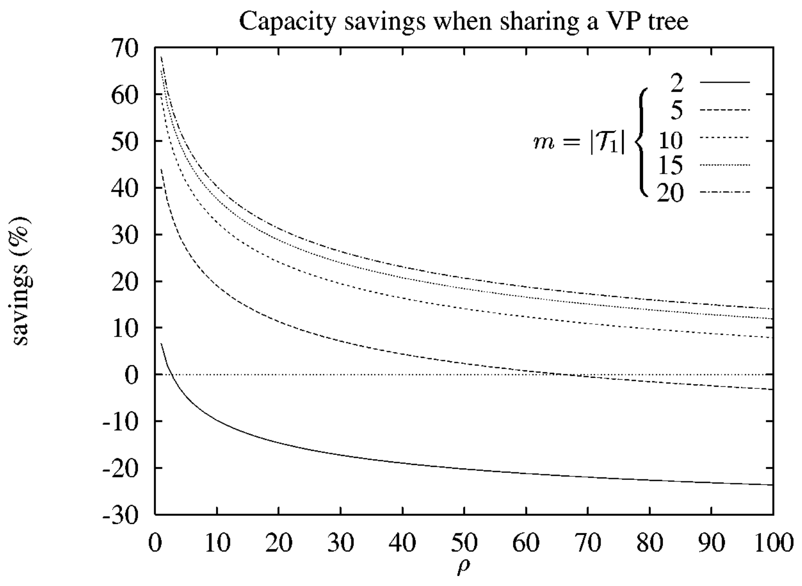

Figure 9. The capacity savings shown are for sharing VP tree $\mathcal{T}_1$ between m multicast groups ($m = |\mathcal{T}_1|$) with destination sets $\mathcal{D}_i \subseteq \mathcal{D}_1$ and tree sizes $|\mathcal{T}_i| = |\mathcal{T}_1| - 1$ for groups $i = 2, 3, \ldots, m$, offered loads $\rho_1 = \rho_2 = \cdots = \rho_m = \rho$, and a blocking probability $B = 0.001$.

larger tree becomes less relative to the total required bandwidth. The incremental gain in savings decreases as the tree sizes grow larger because asymptotically, for fixed offered loads, the savings approaches a constant. For example, in figure 6, the savings for a given ρ and B is $(|\mathcal{T}_1|C_1 + (|\mathcal{T}_1| - 1)C_2 - |\mathcal{T}_1|C_s)/(|\mathcal{T}_1|C_1 + (|\mathcal{T}_1| - 1)C_2)$, where $C_1 = C_2 = \alpha(\rho, B)$, $C_s = \alpha(2\rho, B)$, and α is the inverse Erlang function. As $|\mathcal{T}_1| \to \infty$, the savings approaches $1 - (C_s/(C_1 + C_2))$, i.e., the savings that arises from combining trees of the same size.

From figure 8, we see that aggregation of more and more trees (increasing m) increases savings, but it tapers off. In fact, as m approaches infinity, the savings approaches a constant for fixed offered loads, fixed tree sizes, and constant blocking probability. For the Erlang function, if the offered load and capacity are scaled proportionally, we have that, for $\rho > C$ (heavy traffic)

$$E(m\rho, mC) \overset{m\to\infty}{\longrightarrow} 1 - \frac{C}{\rho} \tag{2}$$

(see [5]). Our scaling is not linear in both ρ and C because as the number of groups (m) grows, the offered load to the shared tree is scaled linearly but the capacity only grows enough to keep the blocking probability constant. However, for large enough m, we are indeed in the heavy traffic or overloaded regime because the blocking probability must remain greater than zero and in the critical and underloaded regimes the blocking probability goes to zero as the capacity grows large [8]. Therefore, letting $\alpha(\rho, B)$ represent the inverse Erlang function, we can use (2) to obtain the rough approximation $\alpha(m\rho, B) \approx m\rho(1 - B)$ for large m. Letting $\rho = \rho_i =$ constant, the savings in figure 8 is $1 - (|\mathcal{T}_1|\alpha(m\rho, B))/(|\mathcal{T}_1|\alpha(\rho, B) + (|\mathcal{T}_1| - 1)(m - 1)\alpha(\rho, B))$ which asymptotically, as $m \to \infty$, becomes

$$1 - \frac{|\mathcal{T}_1|\rho(1 - B)}{(|\mathcal{T}_1| - 1)\alpha(\rho, B)}. \tag{3}$$

Now suppose we combine the last two situations and scale both m and $|\mathcal{T}_1|$ as in figure 9. Using the above approximation, as $m = |\mathcal{T}_1|$ approaches infinity, the savings for fixed $\rho = \rho_i$ is

$$1 - \frac{\rho(1 - B)}{\alpha(\rho, B)}. \tag{4}$$

The second term in (4) can be interpreted as the inverse of the bandwidth required per connection. The bandwidth per connection increases as ρ decreases because of less multiplexing, so for smaller ρ, the inverse of the bandwidth per connection is smaller which leads to higher potential savings. As an example, for $B = 0.001$ and $\rho = 10$, the limit in (4) implies a maximum savings of 52.1%; for $\rho = 100$ the maximum savings is 21.9%. These values are quite reasonable in light of figure 9, where for $m = |\mathcal{T}_1| = 20$ and $\rho = 10$ we have a savings of 40.3%, and for $\rho = 100$ the savings is 14.0%.

Finally, we see from figure 7 that there is a threshold for the smaller tree size below which it never pays to aggregate two candidate trees.

Although the benefits of call level multiplexing are significant, even more capacity savings can be exhibited by incorporating a burst or cell level model. As a simple example, we could model the cell arrival rate of each call by a Gaussian random variable with mean λ and variance σ^2. For a bufferless link with N ongoing connections, it can be shown that the capacity requirement is roughly given by

$$\beta(N) = N\lambda + k\sqrt{N\sigma^2}, \tag{5}$$

where k is a QoS parameter determined by the desired cell loss probability (see, e.g., [13]). For instance, a cell loss probability of 10^{-6} would require $k = 4.75$. Letting $\alpha(\rho, B)$ represent the inverse Erlang function, we need to allocate $\beta(\alpha(\rho, B))$ to satisfy the call level and cell level QoS requirements for fixed cell loss probability and traffic parameters (λ, σ^2).[5] We will explore the impact of the Gaussian traffic model, which is amenable to analysis and often arises as a heavy traffic approximation or aggregation limit, in the simulations of section 3.

3. Heuristics

We now consider the more general problem of having multicast demands from a common source to m destination sets with $\mathcal{D}_i \subset \mathcal{D}_1$ for $i = 2, 3, \ldots, m$. Let $N = |\mathcal{D}_1|$. The subsets $\mathcal{D}_i$, $i = 2, 3, \ldots, m$, can have from 1 to $N - 1$ elements. First, we assume that $\mathcal{D}_m \subset \mathcal{D}_{m-1} \subset \cdots \subset \mathcal{D}_1$, and later we will drop this assumption.

If each set has a known VP tree $\mathcal{T}_i$, the brute force approach to finding a grouping of sets with the least total required capacity is simply to try all possible combinations and keep the best combination. The m sets can be divided into 1 to m groups. The number of possible ways to divide them into k groups is equivalent to finding the number of ways to place m distinguishable balls into k indistinguishable cells such that no cell is empty. This is given by a Stirling number [11] of the second kind $S(m, k)$ where

$$S(m, k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^i \binom{k}{i} (k - i)^m. \tag{6}$$

For each combination, we must find the capacity needed by the k groups, so the computational complexity is proportional to $\sum_{k=1}^{m} kS(m, k)$which grows exponentially. For example, for $m = 2$, $\sum_{k=1}^{m} kS(m, k) = 3$, for $m = 4$, it is 37, and for $m = 8$, it is 17,007. Thus, for reasonably large m, heuristics with polynomial complexity would be preferable to the exponential complexity of the brute force approach.

Although we cannot guarantee optimality, it seems reasonable to try combining pairs of sets starting from the largest set $\mathcal{D}_1$, as suggested by the observation made in section 2 that aggregation of larger trees leads to higher savings. Furthermore, we expect to get the most significant savings from combining trees which are close together in

[5] Note that for multiplexing multiple traffic classes, we need to allocate for the most stringent QoS requirement. See [13] for conditions for which this integration is beneficial.

size. These ideas lead to the following (clustering) algorithm with complexity $\mathrm{O}(m)$. Pseudocode for all the algorithms in this section can be found in [9].

Algorithm 1. Starting with the largest destination set $\mathcal{D}_1$, combine the demands for $\mathcal{D}_1$ and $\mathcal{D}_2$ into a single demand for $\mathcal{D}_1$ if the condition in (1) holds. If successful, try to combine demands for $\mathcal{D}_1$ and $\mathcal{D}_3$, and continue on until unsuccessful with candidates $\mathcal{D}_1$ and $\mathcal{D}_i$, $2 \leqslant i \leqslant m$. If $i < m$, repeat the procedure starting with $\mathcal{D}_i$ and $\mathcal{D}_{i+1}$, continuing on until reaching $\mathcal{D}_m$.

Note that each combination made would result in capacity savings, so regardless of the final grouping, the procedure would be worthwhile, but not necessarily optimal. As a comparison to algorithm 1, we also propose the following algorithm which attempts to make as many combinations as possible with the current destination set before moving on, resulting in an algorithm of complexity $\mathrm{O}(m^2)$.

Algorithm 2. Starting with the largest destination set $\mathcal{D}_1$, combine the demands for $\mathcal{D}_1$ and $\mathcal{D}_2$ into a single demand for $\mathcal{D}_1$ if the condition in (1) holds. Next try to combine demands for $\mathcal{D}_1$ and $\mathcal{D}_3$, and continue on with candidates $\mathcal{D}_1$ and $\mathcal{D}_i$ until $i = m$. If a successful combination was made, repeat the procedure starting with $\mathcal{D}_1$ and $\mathcal{D}_2$ (skipping sets which have previously been absorbed). When a pass without a successful combination has been made, move from $\mathcal{D}_1$ to the next smallest set $\mathcal{D}_j$ that has not been aggregated and repeat from the beginning starting with $\mathcal{D}_j$ and $\mathcal{D}_{j+1}$.

In algorithm 2, we make another pass through the destination sets whenever a successful combination is made. The reason for this is that the offered load of the larger set has been increased and, as can be seen from figure 4, since the threshold is monotonically increasing, the range of offered loads for the smaller tree allowing for capacity savings when sharing has also been increased. Therefore, new combinations could potentially be made that were not allowed on a previous pass through the destination sets.

To see how close to optimal algorithms 1 and 2 might be in practice, we ran simulations of algorithms 1 and 2 and the brute force approach using common random numbers. There were m destination sets with $\mathcal{D}_m \subset \mathcal{D}_{m-1} \subset \cdots \subset \mathcal{D}_1$, and the tree sizes were $1, 2, \ldots, m$, respectively. The offered loads were randomly generated according to a uniform distribution between 0 and $\rho_{\max}$. For each case, the results obtained are 95% confidence intervals based on independent replications. The call blocking probability was held constant at 10^{-3}.

From the results shown in table 1, we see that algorithm 1 is consistently better in total capacity than algorithm 2 and is quite close to the optimal. As expected, the savings percentage drops off significantly as $\rho_{\max}$ grows larger, but it improves as the size and number of the destinations sets m increases. For this scenario, algorithm 1 appears to be a good compromise between complexity and performance.

Table 1
Final capacities and savings percentages for simulations aggregating multicast trees with demands uniformly distributed between 0 and $\rho_{\max}$ and a blocking probability of 10^{-3}.

	Total capacity				Savings percentage		
	Original	Optimal	Alg. 1	Alg. 2	Optimal	Alg. 1	Alg. 2
				$\rho_{\max} = 10$			
$m = 2$	35 ± 6	34 ± 6	34 ± 6	34 ± 6	2 ± 1	2 ± 1	2 ± 1
4	110 ± 17	100 ± 16	100 ± 16	100 ± 16	9 ± 3	9 ± 3	9 ± 3
8	436 ± 50	359 ± 47	363 ± 46	365 ± 47	18 ± 2	17 ± 2	17 ± 3
12	953 ± 109	741 ± 97	747 ± 95	753 ± 92	23 ± 2	22 ± 2	21 ± 1
				$\rho_{\max} = 30$			
$m = 2$	72 ± 11	72 ± 11	72 ± 11	72 ± 11	0.5 ± 0.5	0.5 ± 0.5	0.5 ± 0.5
4	232 ± 39	224 ± 39	224 ± 39	224 ± 39	4 ± 2	4 ± 2	4 ± 2
8	907 ± 120	829 ± 119	830 ± 119	834 ± 116	9 ± 2	9 ± 2	8 ± 2
12	1986 ± 264	1747 ± 251	1769 ± 256	1783 ± 255	12 ± 1	11 ± 2	10 ± 2
				$\rho_{\max} = 50$			
$m = 2$	105 ± 16	105 ± 16	105 ± 16	105 ± 16	0.3 ± 0.3	0.3 ± 0.3	0.3 ± 0.3
4	329 ± 53	322 ± 53	322 ± 53	322 ± 53	2 ± 1	2 ± 1	2 ± 1
8	1327 ± 186	1252 ± 187	1257 ± 186	1258 ± 186	6 ± 1	6 ± 2	5 ± 2
12	2910 ± 408	2665 ± 399	2684 ± 390	2699 ± 388	9 ± 1	8 ± 1	7 ± 1

Two additional statistics are shown in table 2: the link capacity standard deviation and the final number of trees obtained after running the algorithms.[6] Unlike the other statistics, the link capacity standard deviation is topology-dependent, and for our current experiments we used a topology similar to that shown in figure 10 for $m = 5$. The link capacity standard deviation gives us a measure of how well the load is balanced across the links of the network with a standard deviation of zero signifying that all links have the same capacity. As can be seen in table 2, the load is indeed better balanced after running our algorithms, with algorithms 1 and 2 both beating the brute force algorithm. The benefits are most dramatic at low loads and tail off as $\rho_{\max}$ increases. Also, the benefits increase as the number of destination sets m increases. Although it is difficult to quantify, we expect that similar load-balancing benefits might be seen for other more general topologies. The results for the final number of trees are also quite encouraging. Once again, the greatest benefits occur for smaller $\rho_{\max}$, and they grow as m increases. The savings in the number of trees translates to reduced VPI usage and a significant reduction in VP setup and management loads.

We repeated the simulations with the addition of the Gaussian traffic model discussed at the end of section 2. For each connection, the mean and variance of the cell arrival rate was given by $\lambda = 1$ and $\sigma^2 = 1$, respectively. The cell loss probability was held constant at 10^{-6} which translates to a value of 4.75 for the QoS parameter k in (5).

[6] Note that if we tried to optimize in terms of the link capacity standard deviation or the number of trees instead of the total required capacity, we would always end up with one tree.

Table 2
Link capacity standard deviations and final number of trees for simulations aggregating multicast trees with demands uniformly distributed between 0 and $\rho_{\max}$ and a call blocking probability of 10^{-3}.

	Link capacity standard deviation				Final number of trees		
	Original	Optimal	Alg. 1	Alg. 2	Optimal	Alg. 1	Alg. 2
			$\rho_{\max} = 10$				
$m = 2$	7.9 ± 1.6	6.0 ± 2.4	6.0 ± 2.4	6.0 ± 2.4	1.6 ± 0.2	1.6 ± 0.2	1.6 ± 0.2
4	15.3 ± 2.8	10.9 ± 3.8	10.9 ± 3.8	10.9 ± 3.8	2.4 ± 0.5	2.4 ± 0.5	2.4 ± 0.5
8	30.7 ± 3.8	19.0 ± 3.3	15.4 ± 5.3	15.2 ± 5.2	3.4 ± 0.4	3.2 ± 0.5	3.2 ± 0.5
12	46.0 ± 4.1	25.0 ± 3.5	21.5 ± 3.6	21.0 ± 3.6	3.7 ± 0.5	3.4 ± 0.4	3.3 ± 0.5
			$\rho_{\max} = 30$				
$m = 2$	16.3 ± 3.2	15.9 ± 3.5	15.9 ± 3.5	15.9 ± 3.5	1.9 ± 0.1	1.9 ± 0.1	1.9 ± 0.1
4	33.2 ± 6.6	27.3 ± 7.9	26.7 ± 8.5	26.7 ± 8.5	2.8 ± 0.4	2.8 ± 0.4	2.8 ± 0.4
8	64.3 ± 9.1	46.9 ± 8.5	45.8 ± 8.0	45.4 ± 8.0	4.1 ± 0.4	4.1 ± 0.4	4.1 ± 0.4
12	96.4 ± 9.8	70.9 ± 9.3	60.8 ± 9.9	57.8 ± 9.8	5.0 ± 0.3	4.3 ± 0.4	4.2 ± 0.3
			$\rho_{\max} = 50$				
$m = 2$	24.9 ± 5.0	24.3 ± 5.3	24.3 ± 5.3	24.3 ± 5.3	1.9 ± 0.1	1.9 ± 0.1	1.9 ± 0.1
4	47.9 ± 9.0	42.6 ± 9.6	41.4 ± 10.5	41.4 ± 10.5	3.1 ± 0.4	3.0 ± 0.4	3.0 ± 0.4
8	94.5 ± 14.1	79.1 ± 14.0	74.5 ± 12.7	74.5 ± 12.7	4.8 ± 0.5	4.6 ± 0.4	4.6 ± 0.4
12	142 ± 15.0	113 ± 12.5	106 ± 15.4	104 ± 16.1	6.0 ± 0.5	5.4 ± 0.4	5.5 ± 0.4

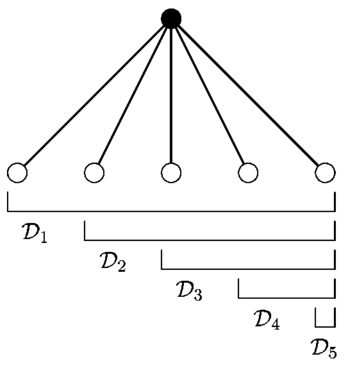

Figure 10. One-level tree topology, shown with $m = 5$, used to determine the link capacity standard deviation in the simulations.

The results, shown in tables 3 and 4, exhibit the same general trends as the previous experiments but with a large increase in the capacity savings percentages. There is also a more significant reduction in the link capacity standard deviation and the final number of trees. Although the Gaussian model is certainly not the best cell level model, it does effectively demonstrate the potential benefits if rate multiplexing is taken into account when aggregating multicast trees.

We shall now drop the assumption that $\mathcal{D}_m \subset \mathcal{D}_{m-1} \subset \cdots \subset \mathcal{D}_1$. We still keep the less restrictive assumption that $\mathcal{D}_i \subset \mathcal{D}_1$ for $i = 2, 3, \ldots, m$, so with $N = |\mathcal{D}_1|$, the subsets $\mathcal{D}_i$, $i = 2, 3, \ldots, m$, can have from 1 to $N - 1$ elements, and we can form

Table 3
Final capacities and savings percentages for simulations aggregating multicast trees with bufferless links, demands uniformly distributed between 0 and $\rho_{\max}$, a call blocking probability of 10^{-3}, a cell loss probability of 10^{-6}, and a Gaussian cell arrival rate with mean 1 and variance 1.

	Total capacity				Savings percentage		
	Original	Optimal	Alg. 1	Alg. 2	Optimal	Alg. 1	Alg. 2
			$\rho_{\max} = 10$				
$m = 2$	86 ± 11	79 ± 12	79 ± 12	79 ± 12	8 ± 3	8 ± 3	8 ± 3
4	265 ± 33	211 ± 29	211 ± 29	211 ± 29	21 ± 4	21 ± 4	21 ± 4
8	1019 ± 85	661 ± 73	664 ± 75	664 ± 75	35 ± 2	35 ± 3	35 ± 3
12	2220 ± 187	1301 ± 138	1326 ± 146	1330 ± 150	42 ± 2	40 ± 2	40 ± 2
			$\rho_{\max} = 30$				
$m = 2$	139 ± 23	135 ± 23	135 ± 23	135 ± 23	3 ± 2	3 ± 2	3 ± 2
4	447 ± 70	389 ± 67	389 ± 67	389 ± 67	13 ± 4	13 ± 4	13 ± 4
8	1740 ± 180	1338 ± 165	1345 ± 168	1350 ± 171	23 ± 2	23 ± 3	23 ± 3
12	3800 ± 395	2703 ± 326	2744 ± 330	2771 ± 340	29 ± 2	28 ± 2	27 ± 2
			$\rho_{\max} = 50$				
$m = 2$	187 ± 31	184 ± 31	184 ± 31	184 ± 31	2 ± 1	2 ± 1	2 ± 1
4	596 ± 101	536 ± 97	536 ± 97	536 ± 97	10 ± 3	10 ± 3	10 ± 3
8	2333 ± 263	1903 ± 247	1929 ± 250	1936 ± 252	19 ± 2	18 ± 2	17 ± 3
12	5098 ± 576	3918 ± 505	3955 ± 505	3981 ± 487	23 ± 2	23 ± 2	22 ± 1

Table 4
Link capacity standard deviations and final number of trees for simulations aggregating multicast trees with bufferless links, demands uniformly distributed between 0 and $\rho_{\max}$, a call blocking probability of 10^{-3}, a cell loss probability of 10^{-6}, and a Gaussian cell arrival rate with mean 1 and variance 1.

	Link capacity standard deviation				Final number of trees		
	Original	Optimal	Alg. 1	Alg. 2	Optimal	Alg. 1	Alg. 2
			$\rho_{\max} = 10$				
$m = 2$	19.0 ± 3.8	3.3 ± 4.9	3.3 ± 4.9	3.3 ± 4.9	1.1 ± 0.2	1.1 ± 0.2	1.1 ± 0.2
4	35.9 ± 5.5	18.3 ± 6.7	16.0 ± 4.9	16.0 ± 4.9	1.9 ± 0.2	1.9 ± 0.2	1.9 ± 0.2
8	70.8 ± 6.6	22.9 ± 8.4	20.5 ± 5.6	20.4 ± 5.6	2.2 ± 0.3	2.1 ± 0.2	2.1 ± 0.2
12	106 ± 7.1	35.7 ± 6.2	18.5 ± 5.2	18.0 ± 5.2	2.7 ± 0.4	2.3 ± 0.4	2.3 ± 0.4
			$\rho_{\max} = 30$				
$m = 2$	30.9 ± 6.1	16.8 ± 10.4	16.8 ± 10.4	16.8 ± 10.4	1.4 ± 0.2	1.4 ± 0.2	1.4 ± 0.2
4	61.3 ± 11.3	40.7 ± 16.6	40.7 ± 16.6	40.7 ± 16.6	2.3 ± 0.5	2.3 ± 0.5	2.3 ± 0.5
8	122 ± 13.8	58.7 ± 19.3	48.0 ± 17.6	47.4 ± 17.0	3.0 ± 0.6	2.7 ± 0.5	2.7 ± 0.5
12	183 ± 14.8	82.5 ± 13.9	65.1 ± 11.5	61.0 ± 11.1	3.1 ± 0.5	2.9 ± 0.4	2.8 ± 0.5
			$\rho_{\max} = 50$				
$m = 2$	42.2 ± 8.2	31.9 ± 12.9	31.9 ± 12.9	31.9 ± 12.9	1.6 ± 0.2	1.6 ± 0.2	1.6 ± 0.2
4	82.2 ± 16.4	58.9 ± 22.3	58.9 ± 22.3	58.9 ± 22.3	2.4 ± 0.5	2.4 ± 0.5	2.4 ± 0.5
8	164 ± 20.1	101 ± 17.3	78.9 ± 27.3	78.2 ± 26.7	3.4 ± 0.4	3.1 ± 0.4	3.1 ± 0.4
12	246 ± 21.6	133 ± 18.3	111 ± 17.0	109 ± 17.0	3.7 ± 0.5	3.2 ± 0.5	3.2 ± 0.5

equivalence classes based on how many elements are in each set. We can also construct relationships based on which destination sets are subsets of other destination sets. Accordingly, we present a modified version of algorithm 1 that proceeds down the hierarchy by equivalence class.

Algorithm 3. Start with the largest destination set $\mathcal{D}_1$ and the equivalence class directly below $\mathcal{D}_1$ with $N - 1$ elements per destination set. For each $\mathcal{D}_i$ in that equivalence class (taken in any order), combine the demands for $\mathcal{D}_1$ and $\mathcal{D}_i$ into a single demand for $\mathcal{D}_1$ if the condition in (1) holds. If successful in combining all members in that equivalence class with $\mathcal{D}_1$ (or if the equivalence class is empty), try to combine demands for $\mathcal{D}_1$ and the members of the equivalence class with $N-2$ elements. Continue on until reaching an equivalence class in which all members are not combined with $\mathcal{D}_1$. Now repeat the procedure starting with each destination set of that class (in any order) and restricting combinations to destination sets which are subsets of the current destination set. Continue on recursively until all destination sets have either been absorbed or have served as the primary candidate to which other destination sets may be combined.

For completeness, we also define a modified version of algorithm 2.

Algorithm 4. Start with the largest destination set $\mathcal{D}_1$ and the equivalence class directly below $\mathcal{D}_1$ with $N - 1$ elements per destination set. For each $\mathcal{D}_i$ in that equivalence class (taken in any order), combine the demands for $\mathcal{D}_1$ and $\mathcal{D}_i$ into a single demand for $\mathcal{D}_1$ if the condition in (1) holds. Next try to combine demands for $\mathcal{D}_1$ and the members of the equivalence class with $N - 2$ elements, and continue on until reaching the equivalence class with the smallest number of elements per destination set. If a successful combination was made, repeat the procedure starting with $\mathcal{D}_1$ and the equivalence class with $N - 1$ elements per destination set (skipping sets which have previously been absorbed). When a pass without a successful combination has been made, move from $\mathcal{D}_1$ to the equivalence class with the largest number of elements per destination set that has members which have not been aggregated, and repeat from the beginning starting with each destination set of that class (in any order) and restricting combinations to destination sets which are subsets of the current destination set. Continue on recursively until all destination sets have either been absorbed or have served as the primary candidate to which other destination sets may be combined.

To simulate algorithms 3 and 4, we began with a destination set of size m and from the $2^m - 2$ proper subsets (excluding the empty set) chose $m - 1$ other destination sets at random at the beginning of each replication. This means that the majority of the destination sets chosen will have close to $m/2$ members. As before, the tree sizes were equal to the destination set sizes, the offered loads were randomly generated according to a uniform distribution between 0 and $\rho_{\max}$, and the results obtained are 95% confidence intervals based on independent replications. To compute the link capacity standard deviation, we used a one-level tree topology, similar to that shown in figure 10,

Table 5
Final capacities and savings percentages for simulations aggregating multicast trees with random destination sets, demands uniformly distributed between 0 and $\rho_{\max}$, and a call blocking probability of 10^{-3}.

	Total capacity				Savings percentage		
	Original	Optimal	Alg. 3	Alg. 4	Optimal	Alg. 3	Alg. 4
				$\rho_{\max} = 10$			
$m = 4$	122 ± 16	112 ± 14	112 ± 14	112 ± 15	7 ± 3	7 ± 3	7 ± 3
8	457 ± 46	404 ± 38	405 ± 38	406 ± 37	11 ± 3	11 ± 3	11 ± 3
12	961 ± 123	844 ± 96	866 ± 93	847 ± 98	12 ± 3	10 ± 4	12 ± 4
				$\rho_{\max} = 30$			
$m = 4$	252 ± 38	246 ± 37	246 ± 37	246 ± 37	3 ± 2	3 ± 2	3 ± 2
8	956 ± 108	916 ± 102	918 ± 102	919 ± 101	4 ± 1	4 ± 1	4 ± 1
12	2004 ± 289	1939 ± 265	1957 ± 265	1939 ± 265	3 ± 1	2 ± 1	3 ± 1
				$\rho_{\max} = 50$			
$m = 4$	372 ± 55	367 ± 55	367 ± 55	367 ± 55	1 ± 1	1 ± 1	1 ± 1
8	1381 ± 161	1351 ± 157	1352 ± 157	1353 ± 156	2 ± 0.9	2 ± 0.9	2 ± 0.9
12	2937 ± 442	2902 ± 428	2912 ± 440	2903 ± 429	1 ± 0.7	0.9 ± 0.5	1 ± 0.7

Table 6
Final capacities and savings percentages for simulations aggregating multicast trees with random destination sets, bufferless links, demands uniformly distributed between 0 and $\rho_{\max}$, a call blocking probability of 10^{-3}, a cell loss probability of 10^{-6}, and a Gaussian cell arrival rate with mean 1 and variance 1.

	Total capacity				Savings percentage		
	Original	Optimal	Alg. 3	Alg. 4	Optimal	Alg. 3	Alg. 4
				$\rho_{\max} = 10$			
$m = 4$	285 ± 30	226 ± 21	226 ± 22	226 ± 21	20 ± 4	20 ± 4	20 ± 4
8	1061 ± 91	728 ± 53	728 ± 53	728 ± 53	31 ± 3	31 ± 3	31 ± 3
12	2242 ± 229	1413 ± 124	1416 ± 127	1416 ± 127	37 ± 2	37 ± 3	37 ± 3
				$\rho_{\max} = 30$			
$m = 4$	486 ± 60	429 ± 51	432 ± 51	431 ± 51	11 ± 3	11 ± 3	11 ± 3
8	1823 ± 183	1505 ± 136	1512 ± 133	1508 ± 135	17 ± 3	17 ± 3	17 ± 3
12	3836 ± 456	3050 ± 319	3100 ± 335	3050 ± 319	20 ± 3	19 ± 5	20 ± 3
				$\rho_{\max} = 50$			
$m = 4$	651 ± 86	597 ± 76	599 ± 76	598 ± 76	8 ± 3	8 ± 3	8 ± 3
8	2448 ± 242	2143 ± 197	2149 ± 195	2150 ± 194	12 ± 3	12 ± 3	12 ± 3
12	5145 ± 652	4461 ± 502	4548 ± 499	4482 ± 511	13 ± 3	11 ± 4	13 ± 4

but now the destination sets are chosen at random from all possible combinations. We performed simulations with and without the Gaussian traffic model using the same traffic parameters and blocking probabilities as before.

The capacity savings results for these experiments are given in tables 5 and 6. See [9] for the results for the link capacity standard deviation and the final number of trees.

We do not repeat the results for $m = 2$ as they are the same as before. Algorithm 3 is close to the optimal in total capacity, but, in contrast to before, it is no longer consistently better than algorithm 4. The savings percentages have been reduced significantly, but they are still quite good when the Gaussian traffic model is included. The results for the link capacity standard deviation and the final number of trees generally exhibit the same trends as before except that the values for algorithm 3 tend to be a bit higher than the other two algorithms. Overall, the impact of including the Gaussian traffic model is more significant than before, and algorithm 3 still appears to be a good compromise between complexity and performance.

4. Role of topology and pre-processing

In this section, we restrict ourselves to the case where the VP multicast layout has not yet been determined, and our procedure is a pre-processing step. As in section 1, we consider the situation with a common source and two destination sets $\mathcal{D}_1$ and $\mathcal{D}_2$ such that $\mathcal{D}_2 \subseteq \mathcal{D}_1$. Suppose that we do not know the actual network topology, but we do know the distances from the source to all destinations in the larger set $\mathcal{D}_1$. The exact ratio $|\mathcal{T}_2|/|\mathcal{T}_1|$ is then unknown, but we can bound it for all possible trees $\mathcal{T}_1$ and $\mathcal{T}_2$ with the specified distances to each destination. In the following, $h(d)$ is the number of hops to destination d.

Lemma 5. Let $u(\mathcal{D}) = \sum_{d\in\mathcal{D}} h(d)$ and $l(\mathcal{D}) = |\mathcal{D}| + \sum_{j=1}^{M-1} \mathbb{1}(j \neq h(d)\ \forall d \in \mathcal{D})$, where $M = \max_{d\in\mathcal{D}} h(d)$ and $\mathbb{1}(\cdot)$ is the indicator function. Then, for any trees $\mathcal{T}_1$ and $\mathcal{T}_2$ satisfying the distance constraints $h(d)\ \forall d \in \mathcal{D}_1$ or $\mathcal{D}_2$, respectively, and connecting a common source to destination sets $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively, with $\mathcal{D}_2 \subseteq \mathcal{D}_1$, we have

$$\frac{l(\mathcal{D}_2)}{u(\mathcal{D}_1)} \leqslant \frac{|\mathcal{T}_2|}{|\mathcal{T}_1|} \leqslant \frac{u(\mathcal{D}_2)}{l(\mathcal{D}_1)}. \tag{7}$$

Proof. To prove the lemma, we show that $l(\mathcal{D}) \leqslant |\mathcal{T}| \leqslant u(\mathcal{D})$ for any tree $\mathcal{T}$ connecting a source to destination set $\mathcal{D}$. The largest number of links occurs when the tree $\mathcal{T}$ has disjoint paths from the source to every destination, i.e., the root of the tree has $|\mathcal{D}|$ branches and no links are shared by two or more source-destination paths. Therefore, $|\mathcal{T}| \leqslant \sum_{d\in\mathcal{D}} h(d) = u(\mathcal{D})$. To establish the lower bound, we assume that no two destinations are collocated. (If not, simply combine the demands for the collocated destinations into a demand for a single combined destination.) First, suppose that $\{h(d) \mid d \in \mathcal{D}\} = \{1, 2, \ldots, M\}$. Then the tree $\mathcal{T}$ with the smallest number of links occurs when the destinations are connected in a straight line. Thus, $|\mathcal{T}| = M = |\mathcal{D}|$. If another destination is added at a duplicate distance $1 \leqslant j \leqslant M$, then another branch must be added originating from a node at distance $j - 1$. If a destination is removed at distance $1 \leqslant j < M$ such that now $j \notin \{h(d) \mid d \in \mathcal{D}\}$, then a branch cannot be removed because the tree would become disconnected. Thus, by construction, $|\mathcal{T}| \geqslant |\mathcal{D}| + \sum_{j=1}^{M-1} \mathbb{1}(j \neq h(d)\ \forall d \in \mathcal{D}) = l(\mathcal{D})$. □

The bounds in lemma 5 are not very tight. For example, define $\Gamma(\mathcal{D}) = \{h(d) \mid d \in \mathcal{D}\}$, and suppose that $\Gamma(\mathcal{D}_2) = \{3, 5\}$ and $\Gamma(\mathcal{D}_1) = \{2, 3, 5\}$. Then we have $5/10 \leqslant |\mathcal{T}_2|/|\mathcal{T}_1| \leqslant 8/5$.

Now consider a similar setup with the additional requirement that $\mathcal{T}_2 \subseteq \mathcal{T}_1$. Tighter bounds than before can be obtained as stated in the following lemma. We use $\mathcal{D}_1 - \mathcal{D}_2$ to designate the set of elements in $\mathcal{D}_1$ that are not in $\mathcal{D}_2$.

Lemma 6. Let $u(\mathcal{D}) = \sum_{d\in\mathcal{D}} h(d)$ and $l(\mathcal{D}) = |\mathcal{D}| + \sum_{j=1}^{M-1} \mathbb{1}(j \neq h(d)\ \forall\ d \in \mathcal{D})$, where $M = \max_{d\in\mathcal{D}} h(d)$ and $\mathbb{1}(\cdot)$ is the indicator function. Then, for any trees $\mathcal{T}_1$ and $\mathcal{T}_2$ satisfying the distance constraints $h(d)\ \forall\ d \in \mathcal{D}_1$ or $\mathcal{D}_2$, respectively, and connecting a common source to destination sets $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively, with $\mathcal{D}_2 \subseteq \mathcal{D}_1$ and $\mathcal{T}_2 \subseteq \mathcal{T}_1$, we have

$$\frac{l(\mathcal{D}_2)}{l(\mathcal{D}_2) + u(\mathcal{D}_1) - u(\mathcal{D}_2)} \leqslant \frac{|\mathcal{T}_2|}{|\mathcal{T}_1|} \leqslant \min\left\{\frac{u(\mathcal{D}_2)}{l(\mathcal{D}_1)}, 1\right\}. \tag{8}$$

Proof. Since $\mathcal{T}_2 \subseteq \mathcal{T}_1$, $|\mathcal{T}_2| \leqslant |\mathcal{T}_1|$, and so $|\mathcal{T}_2|/|\mathcal{T}_1| \leqslant 1$. From lemma 5, we also know that $|\mathcal{T}_2|/|\mathcal{T}_1| \leqslant u(\mathcal{D}_2)/l(\mathcal{D}_1)$, thus establishing the upper bound. For the lower bound, let $\mathcal{T}_{12}$ be a separate tree for $\mathcal{D}_1 - \mathcal{D}_2$. From lemma 5, we know that $|\mathcal{T}_{12}| \leqslant u(\mathcal{D}_1 - \mathcal{D}_2) = u(\mathcal{D}_1) - u(\mathcal{D}_2)$. Since $\mathcal{T}_2 \subseteq \mathcal{T}_1$, $|\mathcal{T}_1| \leqslant |\mathcal{T}_2| + u(\mathcal{D}_1) - u(\mathcal{D}_2)$. So we have

$$\frac{|\mathcal{T}_2|}{|\mathcal{T}_2| + u(\mathcal{D}_1) - u(\mathcal{D}_2)} \leqslant \frac{|\mathcal{T}_2|}{|\mathcal{T}_1|}. \tag{9}$$

To find the lower bound over all trees $\mathcal{T}_2$, we differentiate the left-hand side of (9) with respect to $|\mathcal{T}_2|$. The derivative, $(u(\mathcal{D}_1) - u(\mathcal{D}_2))/(|\mathcal{T}_2| + u(\mathcal{D}_1) - u(\mathcal{D}_2))^2$, is always positive because $u(\mathcal{D}_1) - u(\mathcal{D}_2) \geqslant 0$. Therefore, the left-hand side of (9) is an increasing function of $|\mathcal{T}_2|$. Hence, the smallest possible value of $|\mathcal{T}_2|$, which is $l(\mathcal{D}_2)$, is substituted for $|\mathcal{T}_2|$ to establish the lower bound. $\square$

Using lemma 6, the bounds for our previous example are $5/7 \leqslant |\mathcal{T}_2|/|\mathcal{T}_1| \leqslant 1$, a significant improvement.

By substituting the lower bound of lemma 5 or preferably lemma 6 for $|\mathcal{T}_2|/|\mathcal{T}_1|$ in (1), we can conservatively decide whether or not to combine the demands for $\mathcal{D}_1$ and $\mathcal{D}_2$ into a single demand for the larger set $\mathcal{D}_1$ without knowing the full network topology. As confirmed by lemma 6 and figure 4, large destination sets ($l(\mathcal{D}_2)$ large) with small differences between them ($u(\mathcal{D}_1) - u(\mathcal{D}_2)$ small) will produce a lower bound for $|\mathcal{T}_2|/|\mathcal{T}_1|$ close to 1 (the maximum value) and improve the likelihood of achieving positive capacity savings by combining the demands. Without knowing the distances to the relevant destinations, as we have assumed in this section, it is not feasible to bound $|\mathcal{T}_2|/|\mathcal{T}_1|$ and make any decisions as part of a pre-processing step.

5. Conclusion

In conclusion, we comment on the practicality of the common source assumption for large numbers of destination sets. Instead of an end system, the source could very well be a “core” node inside the network from which core-based trees are established for multicast routing, an architecture being proposed for the Internet [1,2]. A multicast route would consist of a VPC or SVC from the source to the core node followed by the established VP tree. Also, the destinations may be gateway nodes instead of end systems, in which case the VP trees would be entirely contained within the backbone of the network.

It is also worth noting that after aggregation, it may be desirable to perform trunk reservation within a VP tree because of the varying revenues generated by incoming connections [6]. For example, with two types of connections, it is optimal (in terms of total revenue generated) to reserve a certain amount of capacity for the connections which generate more revenue [4].

References

[1] A. Ballardie, Core Based Trees (CBT version 2) multicast routing: Protocol specification, RFC 2189 (September 1997).

[2] A. Ballardie, Core Based Trees (CBT) multicast routing architecture, RFC 2201 (September 1997).

[3] M. Borden, E.S. Crawley, B.S. Davie and S.G. Batsell, Integration of real-time services in an IP-ATM network architecture, RFC 1821 (August 1995).

[4] R.J. Gibbens and F.P. Kelly, Network programming methods for loss networks, IEEE Journal on Selected Areas in Communications 13(7) (September 1995) 1189–1198.

[5] F.P. Kelly, Blocking probabilities in large circuit-switched networks, Advances in Applied Probability 18(2) (June 1986) 473–505.

[6] F.P. Kelly, Routing and capacity allocation in networks with trunk reservation, Mathematics of Operations Research 15(4) (November 1990) 771–793.

[7] S.-B. Kim, An optimal VP-based multicast routing in ATM networks, in: *Proc. IEEE INFOCOM ’96*, Vol. 3 (1996) pp. 1302–1309.

[8] D. Mitra and J.A. Morrison, Erlang capacity and uniform approximations for shared unbuffered resources, IEEE/ACM Transactions on Networking 2(6) (December 1994) 558–570.

[9] M. Montgomery, Managing complexity in large-scale networks via flow and network aggregation, Ph.D. thesis, The University of Texas at Austin (August 1998).

[10] D. Niehaus et al., Performance benchmarking of signaling in ATM networks, IEEE Communications 35(8) (August 1997) 134–143.

[11] F.S. Roberts, *Applied Combinatorics* (Prentice-Hall, Englewood Cliffs, NJ, 1984).

[12] S. Sigarto, Private communication, SBC Technology Resources, Inc. (March 1998).

[13] C.-F. Su and G. de Veciana, On statistical multiplexing, traffic mixes, and VP management, in: *Proc. IEEE INFOCOM ’98*, Vol. 2 (1998) pp. 643–650.

[14] R. Syski, *Introduction to Congestion Theory in Telephone Systems*, Studies in Telecommunication, Vol. 4, 2nd ed. (Elsevier, Amsterdam, 1986).