An Efficient Algorithm for OSPF Subnet Aggregation

Aman Shaikh
Computer Engineering
University of California
Santa Cruz, CA 95064
aman@soe.ucsc.edu

Dongmei Wang, Guangzhi Li, Jennifer Yates, Charles Kalmanek
AT&T Labs (Research)
180, Park Avenue
Florham Park, NJ 07932
{mei,gli,jyates,crk}@research.att.com

Abstract

Multiple addresses within an OSPF area can be aggregated and advertised together to other areas. This process is known as address aggregation and is used to reduce router computational overheads and memory requirements and to reduce the network bandwidth consumed by OSPF messages. The downside of address aggregation is that it leads to information loss and consequently sub-optimal (non-shortest path) routing of data packets. The resulting difference (path selection error) between the length of the actual forwarding path and the shortest path varies between different sources and destinations. This paper proves that the path selection error from any source to any destination can be bounded using only parameters describing the destination area. Based on this, the paper presents an efficient algorithm that generates the minimum number of aggregates subject to a maximum allowed path selection error. A major operational benefit of our algorithm is that network administrators can select aggregates for an area based solely on the topology of the area without worrying about remaining areas of the OSPF network. The other benefit is that the algorithm enables trade-offs between the number of aggregates and the bound on the path selection error. The paper also evaluates the algorithm's performance on random topologies. Our results show that in some cases, the algorithm is capable of reducing the number of aggregates by as much as 50% with only a relatively small introduction of maximum path selection error.

Keywords: OSPF, routing protocol, router configuration management, address aggregation

1. Introduction

Open Shortest Path First (OSPF) [2, 3] is a widely used intra-domain routing protocol in IP, MPLS and optical networks. OSPF is conceptually a link-state routing protocol. In link-state routing protocols, every router acquires a complete view of the network topology. Each link has an associated weight that is administratively assigned. Using the weighted topology, each router computes a shortest path tree with itself as the root [1], and applies the results to build its forwarding table. This assures that packets are forwarded along the shortest paths defined by the link weights to their destinations [2].

For scalability, OSPF allows the network to be divided into areas to define a two-level hierarchy. Area 0, known as the *backbone area*, resides at the top level of the hierarchy and provides connectivity to the non-backbone (non-zero) areas. Figure 1, which is based on a similar figure presented in [4], shows an OSPF network with four areas. In OSPF, each link and subnet are assigned to exactly one area. The routers that have links to multiple areas are called *border routers*. For example, routers A, B, C, D and E are border routers in Figure 1. Every router maintains a separate copy of the topology graph for each area it is connected to. In general, a router does not learn the entire topology of remote areas (i.e., the areas in which the router is not directly connected with), but instead learns the weight of the shortest paths from one or more border routers to each subnet in the remote areas. For example, router A in Figure 1 would learn the entire topology of its attached areas 0 and 3. However, it would only learn distances of various subnets of area 1 from border routers B and C, and of area 2 from border routers D and E. This *summarization* of information outside an area reduces the CPU and memory consumption at routers as well as the network bandwidth consumed by OSPF messages. This makes the protocol more scalable.

OSPF allows border routers to further aggregate a set of subnet addresses into a less specific prefix and advertise a distance to this aggregate prefix instead of distances to the individual subnets. This is referred to as *address aggregation*. Typically, the distance assigned to an aggregate is the maximum of the distance to any subnet covered by the aggregate [2]. Aggregation of subnet addresses into less specific prefixes is controlled by configuration on the border routers. As an example, consider area 1 in Figure 1. Sup-

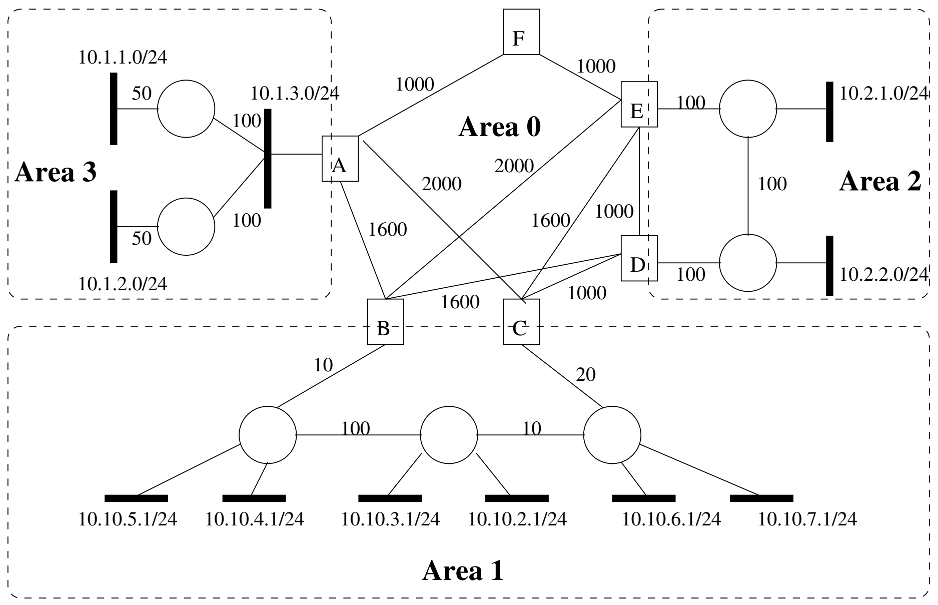

Figure 1. An example OSPF network with four areas. All routers and subnets shown are assumed to be owned by the service provider, and not by the external customers.

pose a single aggregate $10.10.0.0/21$ is used to represent all of the subnets in the area. In that case, router B will advertise a distance of max(10, 110, 120) = 120 to $10.10.0.0/21$, and router C will advertise a distance of max(20, 30, 130) = 130 to $10.10.0.0/21$.

Address aggregation further reduces resource consumption at the routers outside a given area. However, address aggregation generally leads to information loss, which may result in sub-optimal forwarding as the selected path may not be the shortest path to the destination. When a single aggregate $10.10.0.0/21$ is used for representing all subnets in area 1, router A routes all packets via border router B to all destinations in area 1, while routers D and E route all packets via border router C for destinations in area 1. These routing decisions are clearly not optimal (minimum distance). If the routing information in area 1 had not been aggregated, router A, for example, would have routed packets via B if destined for $10.10.5.1/24$ and $10.10.4.1/24$, and via C if destined for $10.10.3.1/24$, $10.10.2.1/24$, $10.10.6.1/24$ and $10.10.7.1/24$. Thus, the aggregation of subnet addresses leads to cases where traffic is not forwarded along the shortest path.

For a given source-destination pair, we define the difference between the length of the selected path and the length of the shortest path to be the *path selection error*. It is of interest to network architects and administrators to know the path selection error introduced by aggregation, and to minimize it. Rastogi *et al.* [4] proposed an algorithm for selecting a given number of aggregates such that the cumulative error in path selection for all *source-destination* pairs is minimized. However, this algorithm requires the knowledge of the entire network topology. In addition, the algorithm computes the aggregates for all areas together, and needs to recompute all the aggregates even if a change is made to a single area. This makes it difficult to use this algorithm for address aggregation in large OSPF networks which tend to undergo constant changes.

The main contribution of this paper is a theorem that proves that the path selection error from any source to any destination can be bounded by a value that depends only on the parameters describing the destination area. More specifically, the theorem proves that the bound on the path selection error can be determined based on three components: the set of border routers in the area, the set of subnets and their distances from the border routers, and the set of aggregates and their distances from the border routers. Using the theorem as a basis, we propose an algorithm that generates a set of aggregates for a given area such that the number of aggregates is minimized subject to a maximum acceptable path selection error. Since the algorithm uses only information about the area of interest, a change to the topology or weights of the area requires a recalculation of aggregates for that area only; not all the areas. For large ISP and enterprise networks, this offers a tremendous advantage over Rastogi's algorithm in terms of network scalability, planning and operations. Another advantage of the algorithm is that it enables a trade-off between the number of aggregates and the bound on the path selection error. The algorithm also opens up the possibility of an on-line implementation where a central server [5] or routers themselves can run the

Table 1. Notation used in this paper

Symbol	Description
s, t	Subnets
$D_s(s,t)$	Shortest path distance from source s to subnet t
X, Y	Aggregates
$D_a(s,t,X)$	Actual distance from s to t on path selected due to X
$E(s,t,X)$	Error in path selection due to the aggregate X, which is $D_a(s,t,X) - D_s(s,t)$
K	Bound on maximum acceptable path selection error
X^F	set of subnets covered by the aggregate X in the area
X_s	Subset of X^F
$F(R_i, X)$	Longest distance from R_i to subnets covered by X, which is $\max_{s \in X^F} D_s(R_i, s)$
B	Number of border routers $(R_1, R, \ldots, R_B)$ in an area
N	Number of subnets $(t_1, t_2, \ldots, t_N)$ in an area
D_{min}	Minimum distance between all (border router, subnet) pairs, which is $min_{1 \le i \le B, 1 \le j \le N} D_s(R_i, t_j)$
D_{max}	Maximum distance between all (border router, subnet) pairs, which is $max_{1 \le i \le B, 1 \le j \le N} D_s(R_i, t_j)$
D_r	Distance range, which is $D_{max} - D_{min}$
A/M	Prefix for a subnet or an aggregate. A is the IP address and M is the mask length.

algorithm and adjust aggregates dynamically as the network topology evolves.

This paper evaluates the performance of the algorithm on randomly generated topologies. Our results show that in some cases, the algorithm is capable of reducing the number of aggregates by as much as 50% with the introduction of only a small path selection error.

The paper is organized as follows: Section 2 formulates the problem. In section 3, we prove the theorem for bounding the path selection error. Section 4 describes the algorithm for determining the set of aggregates. Finally, section 5 evaluates the performance of the algorithm.

2. Problem Formulation

In this section, we formulate the problem underlying our aggregate selection algorithm. We start with the definition of symbols used throughout the paper (see Table 1). We use s and t to denote subnets of an OSPF area, and X and Y to denote the aggregates. Whenever required, we represent addresses of subnets and aggregates as A/M where A represents the IP address, and M represents the mask length. Let $D_s(s,t)$ be the shortest path length from a source s to a destination t. We denote the metric assigned to an aggregate X by a border router R as $F(R,X)$. Furthermore, we denote the length of the selected path from s to t when aggregate X is used to represent t as $D_a(s,t,X)$. Since $D_s(s,t)$ is the length of the shortest path between s and t, $D_a(s,t,X) \ge D_s(s,t)$. We denote the path selection error from s to t due to X as $E(s,t,X)$. Since the path selection error is equal to the difference between the selected path length and the shortest path length, $E(s,t,X) = D_a(s,t,X) - D_s(s,t)$.

We formulate the aggregate selection problem as follows: given an acceptable path selection error, K, a set S of N subnets, a set of B border routers and a B x N matrix representing distances between each (border router, subnet) pair, the objective is to identify a set of aggregates such that the number of aggregates is minimized and the path selection error $E(s,t,X)$ from a source s to any subnet $t \in S$ is bounded by K, i.e., $E(s,t,X) \le K$, where t is covered by the aggregate X.

3. Theorem on Bounding the Path Selection Error

In this section, we prove that an upper bound on the path selection error can be calculated using only local topology information. We start with an example to gain some intuition. Suppose we have two border routers R_1 and R_2, and a subnet t in an area. We assume that s is a source outside the area. Figure 2 represents the shortest path from s to t via R_1 and R_2 as strings with length proportional to the distance of the paths from s. As is evident from Figure 2(a), s will pick the path via R_1 to reach t when no aggregation is used. Now suppose we use an aggregate X to represent t. In this case, whether s selects the path via R_1 or R_2 to reach t, depends on the advertised distances from R_1 and R_2 to X. Since it is likely that both R_1 and R_2 will advertise a distance to X that is no less than the advertised distance to t [2], the paths to X are shown as stretched strings in Figure 2(b) and Figure 2(c). If both strings stretch by the same amount as shown in Figure 2(b), s will still pick R_1 to reach t as shown. In this case, the path selection error is zero. In fact, even if the stretch amount for R_1 is larger than that for R_2, s will still choose R_1 so long as the difference between the stretch amounts for two border routers

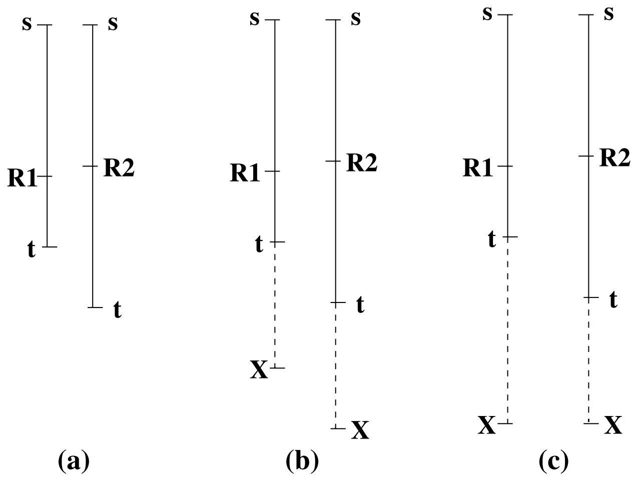

Figure 2. Example showing how path selection error can be bounded with parameters local to the destination area.

is less than the difference in the lengths of two strings in Figure 2(a). Only when the difference between the stretch amounts increases beyond that will s pick R_2 to reach t, and the error in path selection is introduced. Note that the path selection error, which is the difference between the two strings in Figure 2(a), cannot be more than the difference in the stretch amounts of the two strings. In other words, the error in path selection is always bounded by the difference in stretch amounts. Since the stretch amounts of the strings can be determined using parameters of the subnet t's area only, the path selection error can be bounded using the destination area's parameters only. Theorem 1 formalizes this observation.

Theorem 1 *Let us assume that the given area A has B border routers $(R_1, \ldots, R_B)$ and X is one of the aggregates used to cover one or more subnets in area A. Let $F(R_i, X)$ be the metric assigned to aggregate X by R_i. Let t be one of the subnets in A covered by X. Furthermore, let s be a source outside area A. The path selection error from source s to destination t covered by X is bounded by $E(s,t,X) \leq \max_{1 \leq i,j \leq B} |(D_s(R_i,t) - F(R_i,X)) - (D_s(R_j,t) - F(R_j,X))|$.*

Proof: We assume that the shortest path length from s to t via border router R_i is $d_i (1 \leq i \leq B)$ where

$$d_i = D_s(s,R_i) + D_s(R_i,t)$$

Furthermore, we assume that the advertised path length from s to t via R_i due to aggregate X is $c_i (1 \leq i \leq B)$ where

$$c_i = D_s(s,R_i) + F(R_i,X)$$

Let d_k represent the shortest path distance from s to t.

However, since t is covered by the aggregate X, the selected path would be the one with the shortest advertised path length which we denote by c_j. Thus, $c_j \leq c_k$, which is

$$\begin{aligned} D_s(s,R_j) + F(R_j,X) &\leq D_s(s,R_k) + F(R_k,X) \\ D_s(s,R_j) - D_s(s,R_k) &\leq F(R_k,X) - F(R_j,X) \end{aligned}$$

Since the shortest path from s to t is the one passing through router R_k with a distance of d_k, and the selected path is the one passing through router R_j with a distance of d_j, the path selection error $E(s,t,X)$ is:

$$\begin{aligned} &= d_j - d_k \\ &= D_s(s,R_j) + D_s(R_j,t) - D_s(s,R_k) - D_s(R_k,t) \\ &= D_s(s,R_j) - D_s(s,R_k) + D_s(R_j,t) - D_s(R_k,t) \\ &\leq F(R_k,X) - F(R_j,X) + D_s(R_j,t) - D_s(R_k,t) \\ &= (F(R_k,X) - D_s(R_k,t)) - \\ &\quad (F(R_j,X) - D_s(R_j,t)) \\ &\leq \max_{1 \leq i,j \leq B} |(D_s(R_i,t) - F(R_i,X)) - \\ &\qquad (D_s(R_j,t) - F(R_j,X))| \end{aligned}$$ ■

Note that the theorem holds true for an arbitrary cost assignment function $F(R,X)$ as long as all the border routers use the same function. We use this theorem as a basis to design our aggregation algorithm.

4. The Aggregation Algorithm

In this section, we propose an algorithm that determines a set of aggregates that minimizes the number of aggregates required to cover all of the subnets within an area subject to a given bound on path selection error. The algorithm consists of two main steps:

1. Determine a set of candidate aggregates from which aggregates to be advertised by border routers can be selected (see section 4.1).
2. Select a set of aggregates out of the candidate aggregates subject to the error bound (see section 4.2).

Before we describe the algorithm, let us state our assumptions:

1. Subnet addresses in one area do not overlap with those in other areas. This is a reasonable assumption, as network architects typically assign addresses in this manner.
2. The candidate aggregates in one area do not overlap with subnet addresses or candidate aggregates in other areas. The problem of assigning addresses to areas such that there is no overlap across areas is an orthogonal problem, and is beyond the scope of this paper.

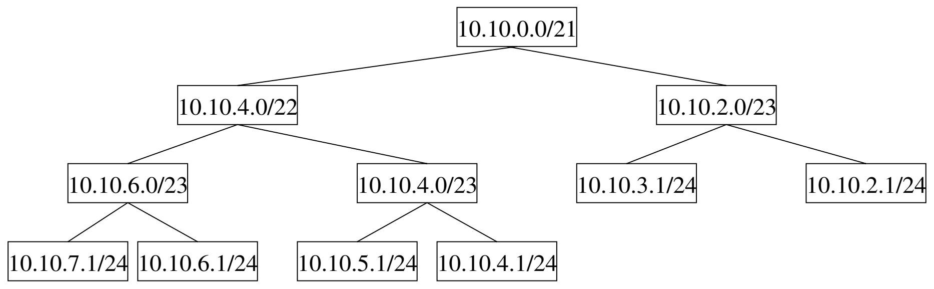

Figure 3. Aggregate Tree T corresponding to area 1 subnets in Figure 1.

4.1. Algorithm for Determining Candidate Aggregates

Procedure 1 $AggrTree(N, \{A_i/M_i \mid 1 \leq i \leq N\})$

```
$r \leftarrow \emptyset$
for $i = 1, \ldots, N$ do
  if $r = \emptyset$ then
    create node $A_i/M_i$ and $r = A_i/M_i$
  else
    $X/m \leftarrow r$
    $Y/k \leftarrow Compare(X/m, A_i/M_i)$
    if $k = m$ then
      $AddChild(X/m, A_i/M_i)$
    else
      create nodes $Y/k$ and $A_i/M_i$
      create an edge between $Y/k$ and $X/m$
      create an edge between $Y/k$ and $A_i/M_i$
      $r \leftarrow Y/k$
    end if
  end if
end for
return $r$
```

Procedure 2 $Compare(A_1/M_1, A_2/M_2)$

```
$Y \leftarrow A_1 \& A_2$, where & is the bit 'and' operator
$M \leftarrow \min(M_1, M_2)$
Let $k$ denote the maximum number of equal bits of $A_1$
and $A_2$ from left to right among the first $M$ bits
return $Y/k$
```

To determine the set of candidate aggregates, we use the concept of an *aggregate tree* proposed by Rastogi *et al.* [4]. An aggregate tree is a binary tree in which each node represents an IP prefix, A/M. Each edge of the tree represents containment relationship between prefixes, i.e., the prefix of the parent node always covers all of the addresses represented by the prefixes of its two children. We build the aggregate tree such that the subnet addresses of an area form the leaves of the tree. Each internal node of the tree represents a candidate aggregate that can be used for representing all of the subnets in its subtree. Figure 3 shows the aggregate tree for the subnets of area 1 in Figure 1.

We now describe the algorithm for building an aggregate tree. To the best of our knowledge, there is no previously published algorithm for building an aggregate tree. Our algorithm starts with an empty tree, and adds one subnet to the tree at a time. At any given time, the partially constructed tree contains candidate aggregates that covers all subnets added to the tree up to that point. The first subnet added to the tree becomes its root. To add each subsequent subnet to the tree, the algorithm starts at the root of the tree. If the root covers the new subnet, the algorithm examines the children of the root. If one of the children covers the subnet, the algorithm further examines the children of this node, and so on until it locates a node P in the tree whose children do not cover the new subnet. Once such a node is located, the algorithm performs a longest prefix match of the subnet with the two children of P. We denote the selected child as H. The algorithm removes the edge between H and its parent, P, and creates two new nodes. The first node represents the new subnet address which we denote as $H1$. The other node represents the most specific prefix that contains both H and $H1$ which we denote as $H2$. $H2$ is then inserted as a child of P, while H and $H1$ become the children of $H2$.

Procedure 1 presents the pseudo-code of $AggrTree$ which implements the above mentioned tree construction algorithm. The procedure takes a set of subnet addresses $\{A_i/M_i \mid 1 \leq i \leq N\}$, and constructs the aggregate tree T. The procedure uses r to represent the root of T. The first prefix, A_1/M_1, initializes the tree by becoming the root. For each subsequent prefix, the procedure tries to determine whether the prefix should be added to the subtree rooted at root r or whether a new root needs to be created. The function $Compare(X/m, A_i/M_i)$ determines the most specific prefix Y/k which covers both X/m and A_i/M_i. Procedure 2 defines the pseudo-code of $Compare(X/m, A_i/M_i)$. Coming back to $AggrTree$, if k is equal to m, root r covers A_i/M_i, and the procedure calls

$AddChild(X/m, A_i/M_i)$ to add A_i/M_i at an appropriate place in the subtree rooted at r. Otherwise, Y/k is made the new root of the tree, and X/m and A_i/M_i are made children of this newly created root.

Procedure 3 $AddChild(X/m, A_i/M_i)$

```
$H_1/h_1$ and $H_2/h_2$ are two children of $X/m$
$Y_1/k_1 \leftarrow Compare(H_1/h_1, A_i/M_i)$
$Y_2/k_2 \leftarrow Compare(H_2/h_2, A_i/M_i)$
if $k_1 > k_2$ then
  if $k_1 = h_1$ then
    $AddChild(H_1/h_1, A_i/M_i)$
  else
    create two nodes $Y_1/k_1$ and $A_i/M_i$,
    delete the edge between $X/m$ and $H_1/h_1$
    create an edge between $X/m$ and $Y_1/k_1$
    create an edge between $Y_1/k_1$ and $H_1/h_1$
    create an edge between $Y_1/k_1$ and $A_i/M_i$
  end if
end if
if $k_2 > k_1$ then
  if $k_2 = h_2$ then
    $AddChild(H_2/h_2, A_i/M_i)$
  else
    create two nodes, $Y_2/k_2$ and $A_i/M_i$
    delete the edge between $X/m$ and $H_2/h_2$
    create an edge between $X/m$ and $Y_2/k_2$
    create an edge between $Y_2/k_2$ and $H_2/h_2$
    create an edge between $Y_2/k_2$ and $A_i/M_i$
  end if
end if
return $X/m$
```

Procedure 3 gives the pseudo-code of $AddChild(X/m, A_i/M_i)$ which calls itself recursively to add the new subnet A_i/M_i as a node in the subtree rooted at X/m. Suppose that H_1/h_1 and H_2/h_2 are two children of X/m, and T_1 and T_2 are the subtrees rooted at these children respectively. The procedure first tries to determine which T_i the subnet A_i/M_i should go in. In order to do so, the procedure calls $Compare$ to determine the most specific prefixes Y_1/k_1 and Y_2/k_2 that covers A_i/M_i and the two children H_1/h_1 and H_2/h_2 respectively. The procedure then picks that subtree T_i for which Y_i/k_i turns out to be more specific. Once the procedure has picked the appropriate subtree, there are two possible cases regarding how A_i/M_i is added to the subtree. In the first case, the root of T_i (H_i/h_i) covers A_i/M_i. Under this case, the procedure calls itself with the appropriate child (H_i/h_i) as the root. In the other case, Y_i/k_i is made the new root of T_i, and H_i/h_i and A_i/M_i become two children of Y_i/k_i.

4.2. Aggregate Selection Algorithm

The aggregate selection algorithm takes an aggregate tree and the acceptable path selection error bound (K) as the input and selects the minimum number of aggregates from the tree such that all of the subnets (i.e., leaves of the tree) are covered and the error bound (K) is satisfied. For efficiency, the algorithm uses a binary search on the number of aggregates, N_a. During the search, for each value of N_a, the algorithm searches the tree to determine whether there is a set of N_a (or $< N_a$) aggregates that satisfies the error bound. The algorithm terminates when the lowest value of N_a is identified.

For a given value of N_a, the algorithm traverses the aggregate tree recursively. It begins at the root of the aggregate tree, with the aim of selecting up to N_a aggregates out of the tree. The algorithm has to consider two options. The first option is to select the root as a candidate aggregate, and try to select up to $N_a - 1$ aggregates from the two subtrees rooted at the children of the root. The other option is to exclude the root from consideration, and select up to N_a aggregates from the two subtrees. Let us denote by N_a' the number of aggregates that the algorithm has to select from the two subtrees. The algorithm recursively looks for up to N_{al} ($\leq N_a'$) aggregates from the left subtree, and up to $N_a' - N_{al}$ aggregates from the right subtree. The value of N_{al} is varied from 0 through to N_a'. For each selected candidate aggregate set, the algorithm calculates the error bound, and compares it with K. The algorithm terminates either when it has identified up to N_a aggregates that satisfy K or when it has failed to identify N_a aggregates by exhausting all possibilities. The algorithm is implemented as a dynamic program so that it solves each subtree problem only once and saves the results to avoid redundant recomputations.

Procedure $FindAggr(X, Y, N_a, K, A(X))$ (see Procedure 4 for the pseudo-code) implements the algorithm described above for a given value of N_a. The procedure tries to determine up to N_a aggregates from the (sub)tree rooted at the aggregate X. Y denotes the most specific aggregate among those selected so far that covers X. Y can be $\emptyset$, and if Y is $\emptyset$, the procedure has to make sure that the selected aggregates cover all of the subnets (i.e., leaves) of the subtree. On the other hand, if Y is not $\emptyset$, the selected aggregates do not have to cover all the subnets since Y represents the aggregate selected higher up the tree, and can cover all the subnets that are not covered by the aggregates selected here. If the aggregates satisfying K are found, the procedure stores them in the set $A(X)$, and returns true; otherwise it returns false with $A(X)$ set to $\emptyset$.

Let us describe $FindAggr$ in more detail. It starts with a simple case of N_a equal to 0. In this case, the procedure returns true with $A(X)$ set to $\emptyset$. If $N_a > 0$, the behavior of the procedure depends on whether there is no selected aggregate Y covering X, i.e., Y is $\emptyset$ or there is a selected aggregate Y covering X, i.e., $Y \neq \emptyset$. Let us focus on the first case here. For this case, if X is a leaf of the tree, it represents a subnet address. Therefore, X must be selected as an aggregate to ensure that the subnet is covered. Since the aggregation error resulting from selecting X is zero, the procedure sets $A(X)$ equal to X, and returns true. If X is not a leaf, then it must have two children which we denote by U and V. At this stage, the procedure considers two options. The first option is to include X in the set of aggregates and try to determine the remaining $N_a - 1$ aggregates from the subtrees rooted at the children of X. The other option is to exclude X, and try to determine all N_a aggregates from the subtrees rooted at the children of X. With both the options, the procedure calls itself recursively to select the remaining aggregates from the subtrees rooted at U and V. Procedure $FindAgg$ calls $MaxErT(X, X^F, K)$ to determine whether using aggregate X for representing subnets in set X^F satisfies the error bound K or not. Notation X^F here represents all the subnets covered by aggregate X in the subtree, and $X^F - A(U) - A(V)$ represents all the subnets covered by X minus those covered by aggregates in $A(U)$ and $A(V)$. The other case where Y is not $\emptyset$ is handled in a similar fashion.

Procedure 5 presents the pseudo-code of $MaxErT(Y, Y_s, K)$. As mentioned earlier, the procedure returns true if error bound calculated according to Theorem 1 for aggregate Y and subnets in Y_s is less than or equal to error bound K; otherwise, it returns false.

4.3. Run-time Analysis of the Algorithm

Having described the algorithm, let us present run-time analysis of the algorithm. First we consider the tree construction algorithm described in section 4.1. Procedure $AddChild$ used for adding a subnet to the aggregate tree can be called $O(h)$ times where h is the height of the tree built so far. An aggregate tree with N leaves has height of $O(\log N)$. Therefore, procedure $AggrTree$ for constructing the aggregate tree takes $O(N \log N)$ time. Next we consider the aggregate selection algorithm described in section 4.2. Since $FindAggr(X, Y, N_a, K, A)$ is implemented as a dynamic program, it is invoked only once for each distinct value of (X, Y, N_a) triplet. The number of distinct (X, Y, N_a) triplets is $(2N-1)(N-1)(N)$ which is $O(N^3)$. Thus, $FindAggr$ can be invoked at most $O(N^3)$ times. The run-time for each invokation of $FindAggr$ depends on three factors: (1) the time taken for the $A(U) \cup A(V)$ operation which can be $O(N^2)$ in the worst-case; (2) the time taken for the $X^F - A(U) - A(V)$ operation which is $O(N^2)$ in the worst-case; and (3) the time taken by $MaxErT$ which can be $O(NB^2)$ in the

Procedure 4 $FindAggr(X, Y, N_a, K, A(X))$

```
if $N_a = 0$, $A(X) \leftarrow \emptyset$, return true
if $N_a > 0$ && $Y = \emptyset$ then
  if $X$ is a leaf, $A(X) \leftarrow X$, return true
  if $N_a = 1$ then
    is $MaxErT(X, X^F, K)$ true ? $\{A(X) \leftarrow X$, return true$\}$ : $\{A(X) \leftarrow \emptyset$, return false$\}$
  else if $N_a > 1$ && $X$ has two children $U$ and $V$ then
    do not select $X$
    for $m = 1, ..., N_a - 1$ do
      if $FindAggr(U, \emptyset, m, K, A(U))$, and $FindAggr(V, \emptyset, N_a - m, K, A(V))$ both are true, then $A(X) \leftarrow A(U) \cup A(V)$, return true
    end for
    select $X$
    for $m = 0, ..., N_a - 1$ do
      $X^F$ = set of subnets covered by $X$
      if $FindAggr(U, X, m, K, A(U))$, $FindAggr(V, X, N_a - 1 - m, K, A(V))$, and $MaxErT(X, (X^F - A(U) - A(V)), K)$ all are true, then $A(X) \leftarrow X \cup A(U) \cup A(V)$, return true
    end for
    $A(X) \leftarrow \emptyset$, return false
  end if
end if
if $N_a > 0$ && $Y \neq \emptyset$ then
  if $X$ is a leaf $A(X) \leftarrow X$, return true
  if $X$ is not a leaf then
    $X$ has two children $U$ and $V$
    do not select $X$
    for $m = 0, ..., N_a$ do
      $X^F$ = set of subnets covered by $X$
      if $FindAggr(U, Y, m, K, A(U))$, $FindAggr(V, Y, N_a - m, K, A(V))$, and $MaxErT(Y, (X^F - A(U) - A(V)), K)$ are true, then $A(X) \leftarrow A(U) \cup A(V)$, return true
    end for
    select $X$
    for $m = 0, ..., N_a - 1$ do
      $X^F$ = set of subnets covered by $X$
      if $FindAggr(U, X, m, K, A(U))$, $FindAggr(V, X, N_a - 1 - m, K, A(V))$, and $MaxErT(X, X^F - A(U) - A(V), K)$ all are true, then $A(X) \leftarrow X \cup A(U) \cup A(V)$, return true
    end for
  end if
end if
$A(X) \leftarrow \emptyset$, return false
```

Procedure 5 $MaxErT(Y, Y_s, K)$

```
for $i = 1, ..., |Y_s|$ do
    if $\max_{1 \leq u,v \leq B} |(D_s(R_u, y_i)) - F(R_u, Y)) - (D_s(R_v, y_i)) - F(R_v, Y))| \geq K$ then
        return false
    end if
end for
return true
```

worst-case. Since each of these three operations are invoked in a for-loop which runs $O(N)$ times, each invokation of $FindAggr$ can take $O(N^3 + N^2B^2)$ time. Therefore, the overall run-time complexity of $FindAggr$ can be bounded as $O(N^6 + N^5B^2)$.

5. Performance Evaluation

This section evaluates the performance of our aggregation algorithm. We start by characterizing how effective the algorithm is in reducing the number of aggregates as the error bound is increased. The effectiveness of the algorithm in this regard depends on three factors: (1) the number of subnets in an area and the address structure of the subnets, (2) the number of border routers in the area, and (3) the distances of border routers to the subnets. Our evaluation characterizes the effect of all of these factors on the performace of the algorithm. After that, we characterize the run-time of the algorithm and show how it scales as the number of subnets increases.

For performance evaluation, we have implemented the algorithm in C++. We use a single OSPF area as an input to the algorithm. In the absence of any realistic data to use for OSPF areas, we synthetically generated the required parameters of an area, namely, a set of border routers, a set of subnets and a B x N distance matrix representing distances between each (border router, subnet) pair. We randomly selected both the subnet addresses and each element of the distance matrix. Each subnet was assigned an address block with a randomly generated mask length between 24 and 30. The addresses themselves were assigned as compactly as possible. For example, the address assignment for four subnets looks like this: 10.0.0.0/26, 10.0.0.64/28, 10.0.0.96/27 and 10.0.0.128/25. Distances between each (border router, subnet) pair was randomly selected between D_{min} and D_{max}. Values of D_{min} and D_{max} themselves are not important for our purposes; what matters is the difference between these values which we denote as D_r.

The first set of results show how the number of aggregates decreases as the error bound is increased for a given value of (N, B, D_r) triplet. We collected ten samples for each value of (N, B, D_r) with different seed values being used for each sample. Our results present the mean of

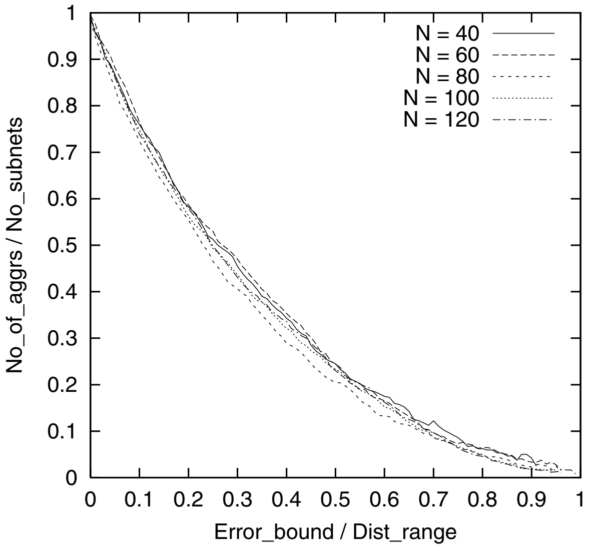

Figure 4. Number of aggregates versus error bound for varying number of subnets (B = 2 and D_r = 100).

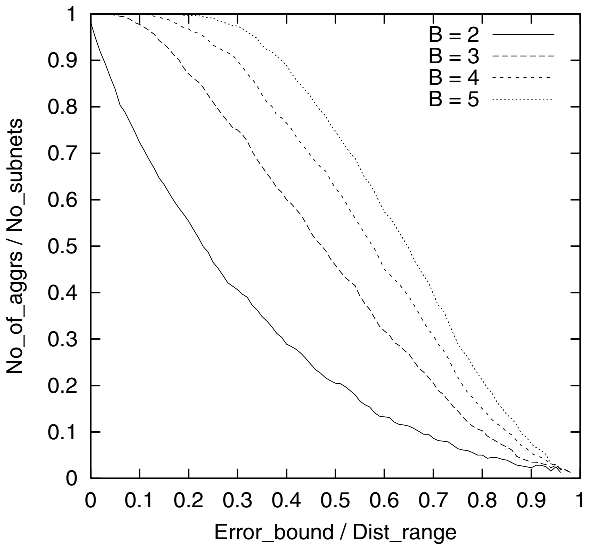

Figure 5. Number of aggregates versus error bound for varying number of border routers (N = 80 and D_r = 100).

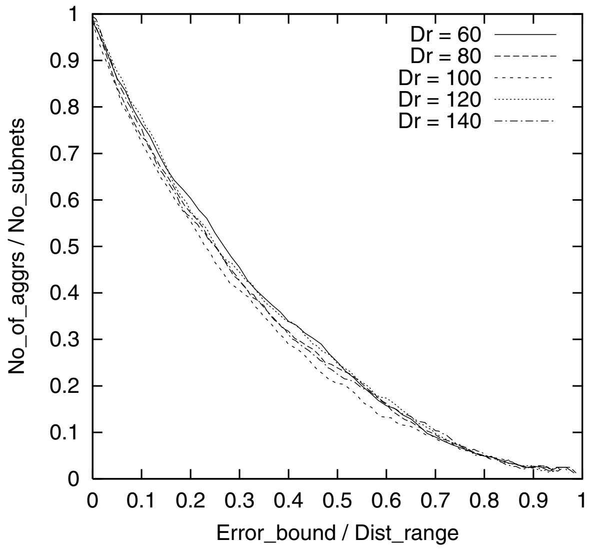

Figure 6. Number of aggregates versus error bound for varying D_r (N = 80 and B = 2).

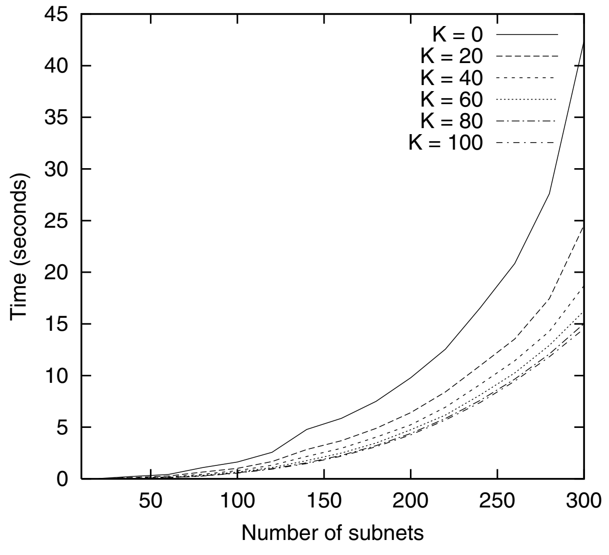

Figure 7. Run-time of aggregate selection algorithm versus the number of subnets (B = 2 and D_r = 100).

these ten samples. In the plots that follow we normalize the number of aggregates by dividing it by the number of subnets, and normalize the error bound by dividing it with the distance range, D_r. Figures 4, 5 and 6 plot the aggregation ratio versus error bound ratio for different numbers of subnets, different numbers of border routers, and different values of the distance range respectively.

Figure 4 demonstrates that the algorithm is effective in reducing the number of aggregates since it is able to reduce the number of aggregates by 50% for an introduction of 25% in the error bound ratio. This shows that significant aggregation can be achieved while introducing only a relatively small path selection error. The figure also shows that the number of subnets (N) has no impact on the characteristic of the error bound ratio versus aggregation error curve.

Figure 5 demonstrates that the performance of the aggregation algorithm is sensitive to the number of border routers. As we increase the number of border routers, the aggregation algorithm becomes less effective. From Theorem 1, we see that the error bound is given by $E(s,t,X) \leq \max_{1\leq i,j\leq B} |(D_s(R_i,t) - F(R_i,X)) - (D_s(R_j,t) - F(R_j,X))|$. Thus, the bound is likely to increase as the number of border routers increases especially for scenarios where the distances are randomly selected. This figure shows that our algrothm is more effective at reducing the number of aggregates when the number of border routers is small.

Figure 6 shows the impact of distance range (D_r) on the characteristic of the error bound ratio versus aggregation error curve. The number of subnets was set to 80 and the number of border routers was set to two for this figure. As the figure shows, the performance of the aggregation algorithm is independent of the distance range after we normalize the number of aggregates and the error bound.

Next we look at the run-time characterstic of the algorithm. We only report the run-time of the aggregate selection algorithm and exclude the run-time of the tree construction time. We do so for two reasons. The first reason is that the time spent within the aggregate selection procedures dominates the overall execution time. The other reason is that the tree construction needs to be executed only when the subnet addresses change whereas the aggregate selection algorithm needs to be executed whenever the distance matrix changes or the user wants to change the error bound. We believe that in operational environments, the distance matrices change much more often than the subnet addresses. Figure 7 plots the run-time versus the number of subnets for different error bounds. The number of border routers was set to two and the distance range was set to 100 for this plot. As expected, as the error bound gets tighter, it becomes more difficult to select the desired aggregates, and so the run-time of the algorithm increases. We have determined that the curves in Figure 7 lie between $O(N^3)$ and $O(N^4)$, which is much better than the worst-case run-time complexity ($O(N^6 + N^5B^2)$) derived in section 4.3. The worst-case complexity was derived with the assumption that the algorithm has to invoke $FindAggr$ for all possible values of (X, Y, N). This does not hold true for the results presented here. We have verified that the algorithm was

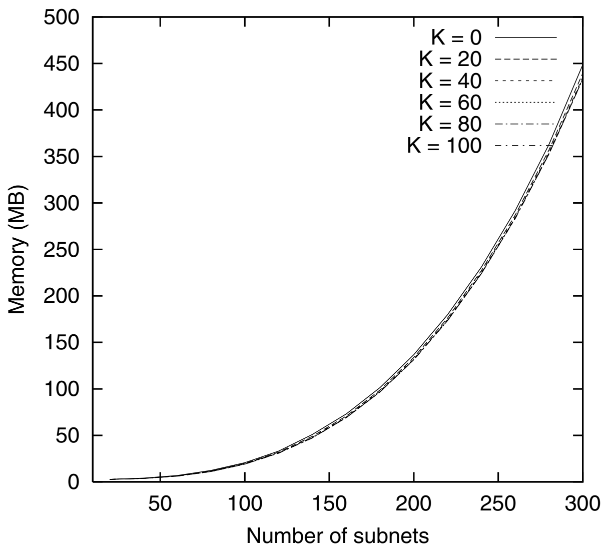

Figure 8. Total memory used by the aggregate selection algorithm versus the number of subnets (B = 2 and D_r = 100).

able to find the minimum number of aggregates by exploring only a part of the search space. We also measured the run-time for varying number of border routers, and found that the number of border routers did not have any effect on the run-time.

Finally, Figure 8 plots the maximum memory consumption of the algorithm versus the number of subnets for different error bounds. In our implementation, the majority of the memory is used for a three-dimensional array that implements the dynamic programming by keeping track of the aggregates as they are found for a given value of the (X, Y, N) triplet. Since the size of the array has $O(N^3)$ scaling, the overall space complexity is also $O(N^3)$. However, most space reserved for the array is never utilized since the algorithm typically converges to the final result after exploring only a very small portion of the search space. For example, only 1% of the reserved array memory is utilized when N is 300 and K is set to zero. Thus, for large networks, the memory requirements of the algorithm could be reduced by simply using more efficient data structures.

6. Conclusions

Address aggregation is used for reducing resource consumption on routers outside an OSPF area. However, address aggregation leads to suboptimal routing since a path selected between a source outside an area and an aggregated destination in the area can deviate from the shortest path. This paper proved that the path selection error introduced by aggregation can be bounded by the parameters associated with the destination area only. Based on this theoretical result, we proposed an algorithm for determining the minimum number of aggregates subject to a user-defined bound on the acceptable maximum error in path selection. The algorithm can be applied on a per-area basis, which is a tremendous advantage from both the operational and scalability perspectives. The algorithm also offers network administrators the ability to trade off the number of aggregates with the acceptable path selection error. Another benefit of the algorithm is that it is amenable to an on-line implementation on border routers themselves or a central server [5].

We evaluated the effectiveness of the algorithm on randomly generated OSPF areas. The simulation results demonstrated that the algorithm is able to significantly reduce the number of aggregates (50%) while introducing only a relatively small error bound (25%) for two border routers. However, as the number of border routers increases, the effectiveness of the algorithm deteriorates. We plan to address this as a part of our future work. We are also looking into ways of reducing overall time and space complexity of the algorithm. We also plan to evaluate the effectiveness of the algorithm on realistic OSPF area topologies. Finally, we plan to look at ways of assigning addresses to subnets belonging to a given area in a manner that increases the effectiveness of our aggregate selection algorithm.

Acknowledgments

We would like to thank Rick Greer, Fred True, Joel Gottlieb and Don Caldwell for their help in measuring the memory consumption of our algorithm. We would also like to thank the anonymous reviewers for their feedback and comments.

References

[1] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik 1*, pages 269–271, 1959.

[2] J. T. Moy. *OSPF : Anatomy of an Internet Routing Protocol*. Addison-Wesley, January 1998.

[3] J. T. Moy. OSPF Version 2. RFC2328, April 1998.

[4] R. Rastogi, Y. Breitbart, M. Garofalakis, and A. Kumar. Optimal Configuration of OSPF Aggregates. In *Proc. IEEE INFOCOM*, June 2002.

[5] A. Shaikh, M. Goyal, A. Greenberg, R. Rajan, and K. Ramakrishnan. An OSPF Topology Server: Design and Evaluation. *IEEE Journal on Selected Areas in Communications (J-SAC)*, 20(4), May 2002.