An Approach to Hierarchical Inter-Domain Routing with On-Demand ToS and Policy Resolution*

Cengiz Alaettinoğlu, A. Udaya Shankar
Department of Computer Science
University of Maryland
College Park, Maryland 20742
{ca,shankar}@cs.umd.edu

Abstract

Traditional inter-domain routing protocols based on superdomains maintain either "strong" or "weak" ToS and policy constraints for each visible superdomain. With strong constraints, a valid path may not be found even though one exists; with weak constraints, an invalid domain-level path may be treated as a valid path.

We present an inter-domain routing protocol based on superdomains, which always finds a valid path if one exists. Both strong and weak constraints are maintained for each visible superdomain. If the strong constraints of superdomains on a path are satisfied, then the path is valid. If the weak constraints of a superdomain are satisfied but the strong constraints are not, the source uses a query protocol to obtain a more detailed "internal" view of the superdomain, and searches again for a valid path. Our protocol handles topology changes, including node/link failures that partition superdomains.

1 Introduction

A computer internetwork, such as the Internet, is an interconnection of backbone networks, regional networks, metropolitan area networks, and stub networks (campus networks, office networks and other small networks). Stub networks are the producers and consumers of the internetwork traffic, while backbones, regionals, and MANs are transit networks. (Most of the networks in an internetwork are stub networks.) Each network consists of nodes (hosts, routers) and links. Two networks are ***neighbors*** when there is one or more links between nodes in the two networks (see Figure 1).

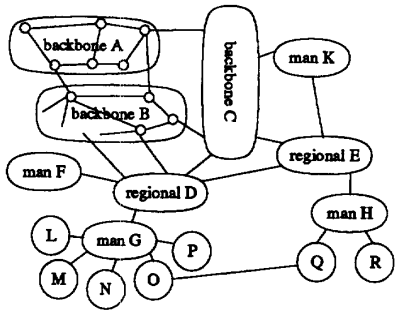

Figure 1: A portion of an internetwork. (Circles represent stub networks.)

An internetwork is organized into *domains*. A domain is a set of networks (possibly consisting of only one network) administered by the same agency. Within each domain, an ***intra-domain routing protocol*** is executed that provides routes between source and destination nodes in the domain. This protocol can be any of the typical ones, i.e., next-hop or source routes computed using distance-vector or link-state algorithms. To offer different *type-of-service* (ToS) constraints of applications (e.g. low delay, high throughput, high reliability, minimum monetary cost), each node maintains a *cost* for each outgoing link[1]. The intra-domain routing protocol should choose optimal paths based on these costs.

*This work is supported in part by RL and DARPA under contract F30602-90-C-0010 to UMIACS at the University of Maryland, and by National Science Foundation Grant No. NCR 89-04590. The views in this report are those of the author(s) and should not be interpreted as representing the official policies of DARPA, RL, or the U.S. Government. Computer facilities were provided in part by NSF grant CCR-8811954.

[1] The cost of a link equals a vector of values if the link is up; each cost value indicates how expensive it is to cross the link according to some ToS constraint such as delay, throughput, reliability, etc. The cost of a link equals ∞ if the link is down.

0-8186-3670-X/93 $03.00

Across all domains, an *inter-domain routing protocol* is executed that provides routes between source and destination nodes in different domains. This protocol must satisfy various constraints:

- It must satisfy *policy constraints*, which are administrative restrictions on the inter-domain traffic [6, 7, 4]. Policy constraints are of two types: *transit policies* and *source policies*. The transit policies of a domain U specify how other domains can use the resources of U (e.g. \$0.01 per packet, no traffic from domain V). The source policies of a domain U specify constraints on traffic originating from U (e.g. domains to avoid/prefer, acceptable connection cost). Transit policies of a domain are public (i.e. available to other domains), whereas source policies are usually private.
- An inter-domain routing protocol must also satisfy ToS constraints of applications. To do this, it must keep track of the types of services offered by each domain [4].
- An inter-domain routing protocol must handle arbitrary interconnection of networks [7] (e.g. we do not want to hand-configure special routes as "backdoors").
- Inter-domain routing protocols must scale up to very large internetworks, i.e. with a very large number of domains. Practically this means that processing, memory and communication requirements should be much less than linear in the number of domains.
- Inter-domain routing protocols must automatically adapt to link cost changes and node/link failures and repairs, including failures that partition domains [11].

A straight-forward approach to inter-domain routing is domain-level source routing with link-state approach [6, 4]. In this approach, each router[2] maintains a *domain-level view* of the internetwork, i.e., a graph with a vertex for every domain and an edge between every two neighbor domains. Policy and ToS information is attached to the vertices and the edges of the view.

When a source node needs to reach a destination node, it (or a router in the source's domain) first examines this view and determines a *domain-level source route* satisfying ToS and policy constraints, i.e., a sequence of domain ids starting from the source's domain and ending with the destination's domain. Then, the packets are routed to the destination using this domain-level source route and the intra-domain routing protocols of the domains crossed. That is, a router uses its intra-domain routing protocol to forward a packet towards the next domain in the packet's domain-level source route. This means that the intra-domain routing tables must also contain a route (next-hop or source route) for every neighboring domain.

The disadvantage of this straightforward scheme is that it does not scale up for large internetworks. The storage at each router is proportional to $N_D \times E_D$, where N_D is the number of domains and E_D is the average number of neighbor domains to a domain. The communication cost is proportional to $N_R \times E_R$, where N_R is the number of routers in the internetwork and E_R is the average router neighbors of a router (topology changes are flooded to all routers in the internetwork).

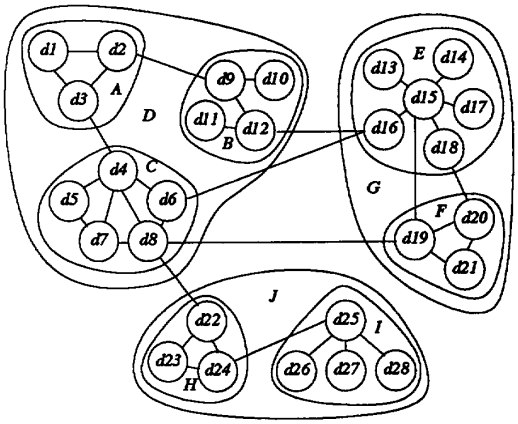

Figure 2: An example of superdomain hierarchy.

To achieve scaling, several approaches based on aggregating domains into superdomains have been proposed [10, 12, 5]. Here, domains are grouped hierarchically into *superdomains*: each domain is a level 1 superdomain, "close" level 1 superdomains are grouped into level 2 superdomains, "close" level 2 superdomains are grouped into level 3 superdomains, and so on (see Figure 2). Each router x maintains a view that contains the level 1 superdomains in x's level 2 superdomain, the level 2 superdomains in x's level 3 superdomain (excluding the x's level 2 superdomain), and so on. Thus a router maintains a smaller view than it would in the absence of hierarchy. For the superdomain hierarchy of Figure 2, the views of two routers (one in domain $d1$ and one in domain $d16$) are shown in Figures 3 and 4.

The superdomain approach has several drawbacks. One drawback is that the aggregation results in loss of

[2] Not all nodes maintain routing tables. A router is a node that maintains a routing table.

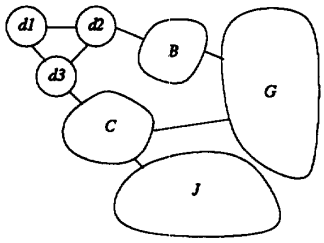

Figure 3: View of a router in $d1$.

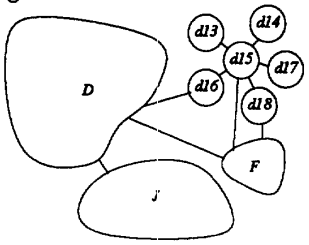

Figure 4: View of a router in $d16$.

domain-level ToS and policy information. A superdomain is usually characterized by a single set of ToS and policy constraints. Nodes outside the superdomain assume that this set of constraints applies uniformly to each of its children (and by recursion to each domain in the superdomain). If there are children superdomains with different (possibly contradictory) constraints in a superdomain, then there is no good way of deriving the ToS and policy constraints of the superdomain. The usual techniques [10] are to obtain either a *strong* set of constraints or a *weak* set of constraints[3]. If strong (weak) constraints are used for policies, the superdomain is assumed to enforce a policy constraint if that policy constraint is enforced by some (all) of its children. If strong (weak) constraints are used for ToS constraints, the superdomain is assumed to support a ToS if that ToS is supported by all (some) of its children.

Both techniques have problems. With the strong constraints, a valid path may not be found even though one exists. With the weak constraints, an invalid domain-level path may be treated as a valid path. For the example in Figure 2, suppose $d16$ allows transit traffic from $d1$ while $d19$ does not. If the constraints of E, F and G are obtained by strong version, then G would not allow transit traffic from $d1$, even though there are transit paths through $d16$ and avoiding $d19$. If the weak version is used, G allows transit traffic from $d1$, forcing $d19$ to allow transit traffic from $d1$.

Another problem with superdomains is that even if a valid superdomain-level path is determined for an application, it is not straight-forward to implement that in terms of superdomain-level source routes. This is because routers in different superdomains maintain views of different sets of superdomains. Thus, a superdomain-level source route can be meaningless at some intermediate superdomain's router x because the next superdomain in the source route is not visible to x. For example, superdomain-level source route $\langle d2, B, G, C\rangle$ created at a router in $d2$ becomes meaningless once the packet is in G, where C is not visible.

Node/link failures can partition superdomains at various levels. The varying visibility of routers complicates the handling of such changes.

Our Contribution

In this paper, we present an inter-domain routing protocol based on superdomains, which always finds a valid path if one exists. Both strong and weak constraints are maintained for each visible superdomain. If the strong constraints of superdomains on a path are satisfied, then the path is valid. If the weak constraints of a superdomain are satisfied but the strong constraints are not, the source uses a query protocol to obtain a more detailed "internal" view of the superdomain, and searches again for a valid path.

We overcome the superdomain-level source routing problem by having the source insert exit and entry domains whenever the next superdomain in the source route is not visible to routers in the previous superdomain.

We use a link-state view update protocol to handle topology changes including failures that partition superdomains at any level.

Several approaches have been proposed based on the superdomain hierarchy [12, 10, 8, 5, 14] and landmark hierarchy [13]. In each case, the approach either suffers from the loss of ToS and policy information (and hence not find a valid path which exists), or it is still in a preliminary stage. In our opinion, solving this problem without loosing scalability requires a query mechanism to obtain ToS and policy details wherever it is needed to find a valid path.

In [2], we introduced a new *viewserver* hierarchy. Here, special routers called viewservers maintain the view of domains in a surrounding precinct. Viewservers are organized hierarchically such that for each viewserver, there is a domain of a lower level viewserver in its view, and views of top level viewservers includes domains of other top level

[3] "strong" and "weak" are referred to as "union" and "intersection" in [10]

viewservers. An addressing and route discovery schemes are introduced. Even though this approach does not use superdomains, it scales well to large internetworks and does not lose detail ToS and policy information.

Organization of the paper

Section 2 describes the mapping of links to superdomain-level edges and the superdomain-level edges visible to routers. Section 3 introduces exit and entry domain attributes for superdomain-level edges as a way to achieve superdomain-level source routing using intra-domain routing service. Section 4 introduces the update mechanism by which routers maintain superdomain-level edges and their attributes. Routers exchange "higher resolution" superdomain-level edges referred to as raw-edges. Section 5 introduces the query protocol used by a source router to obtain a valid path. Section 6 shows how to modify the results of Sections 2-5 to handle topology changes including those that cause partitions. In Section 7, we give concluding remarks.

For precise specification of the protocols in this paper, the reader is referred to [1].

2 Superdomain-level Edges

The superdomain hierarchy defines the following parent-child relationship: a level i super domain is the parent of each level $i-1$ superdomain it contains. Note that top level superdomains have no parents, and level 1 superdomains, which are just domains, have no children. Let *ancestor*, *descendant*, and *sibling* have the usual meanings in the parent-child relationship[4].

Each router maintains a view of a subset of superdomains, referred to as its visible superdomains. The *visible superdomains* of a router x are (1) x's domain itself, (2) siblings of x's domain, and (3) siblings of ancestors of x's domain. Note that if a superdomain U is visible to a router, then no ancestor or descendant of U is visible to the router. In Figure 2, the visible superdomains of a router in $d1$ are $d1, d2, d3, B, C, G, J$.

We say that (d_i, d_j), where d_i and d_j are domain ids, is a *d-edge* (for "domain-level" edge) iff there is one or more links from nodes in d_i to nodes in d_j. We say that (U, V), where U and V are superdomain ids, is an *sd-edge* (for "superdomain-level" edge) iff there is one or more d-edges from domains in U to domains in V. Each of the d-edges is said to *participate* in the sd-edge. Two superdomains (domains) are *neighbors* if there is an sd-edge (d-edge) between them.

Note that a d-edge can participate in many sd-edges. Specifically, a d-edge (d_i, d_j) participates in sd-edge (U, V) for every ancestor U of d_i and every ancestor V of d_j such that $U \neq V$. However, to a router at most one of these sd-edges is visible; namely the (U, V) edge where U and V are visible to the router. None of the sd-edges are visible to the router if the d-edge is completely inside a visible superdomain. For example in Figure 2, the d-edge $(d6, d16)$ participates in the sd-edges $(d6, d16)$, $(d6, E)$, $(d6, G)$, $(C, d16)$, (C, E), (C, G), $(D, d16)$, (D, E), and (D, G); of these edges, only $(d6, G)$ is visible to a router in $d6$, and only (C, G) is visible to a router in $d1$.

The view maintained by a router consists of the visible superdomains and the visible sd-edges. Strong and weak constraints are associated with each superdomain and each sd-edge. In addition a cost is associated with each sd-edge. The cost of an sd-edge (U, V) equals a vector of values if the sd-edge is up; each cost value indicates how expensive it is to cross the sd-edge according to some ToS constraint. The cost equals ∞ if all links from U to V are down.

Each cost value of an sd-edge (U, V) can be derived from the cost values of the links in U and links from U to V [3]. For example, if U and V are domains, the cost of the d-edge (U, V) can be calculated as the maximum/average cost of the routes from any router in U to the first router in V. For higher level superdomains U and V, cost of the sd-edge (U, V) can be derived from the costs of sd-edges between the children of U and from the costs of sd-edges from the children of U to the children of V.

3 Addressing and Superdomain-Level Source Routing

A node h in domain d is totally identified by $d{:}h$. But a router cannot use this id to route packets destined for the node unless d is a visible domain of the router. For routing purposes, each domain (and node) has an address, defined by concatenating the superdomain ids starting from the top level and going down to the domain (node). For example in Figure 2, the address of domain $d16$ is $G.E.d16$, and the address of a node h in $d16$ is $G.E.d16.h$.

We assume that each router's intra-domain routing table contains a route (e.g. next-hop node) for

[4] A superdomain X is an ancestor (descendant) of a superdomain Y iff X is Y's parent (child) or X is an ancestor (descendant) of Y's parent (child). X is a sibling of Y iff X and Y have the same parent and $X \neq Y$.

(1) each node in the domain, (2) each neighbor domain (whether or not it is visible), and (3) each visible superdomain (obtaining this route is described later).

Each packet contains its destination's address. To a router, only one of the superdomains in the destination address is visible. The router forwards the packet using its route for this visible superdomain. For example, in figure 2, to forward a packet destined to $d16$, a router in $d1$ uses its route for G.

However, routing based only on the destination address may not provide a valid path. It may be necessary to follow a superdomain-level source route[5]. But superdomain-level source routing is not straight-forward. As we saw in the Introduction, a superdomain-level source route created at one router can become meaningless at an intermediate domain's router, because the next superdomain in the source route is not visible.

We propose a simple solution to this. In a superdomain-level source route $\langle U_1, U_2, \ldots, U_n \rangle$, for every superdomain U_{i+1} that is not visible to the routers of U_i, the source inserts between U_i and U_{i+1} the address of a domain d in U_i and the id of a domain e in U_{i+1} such that d and e are neighbors. We refer to d as an *exit domain* (of U_i), and to e as an *entry domain* (of U_{i+1}). For example in Figure 3, suppose a router x in $d2$ chooses the superdomain-level source route $\langle d2, B, G, C \rangle$ to some node in C. Because C is not visible to a router in G (see Figure 4), x inserts $G.E.d16$ and $d6$ between G and C in the source route. In G, packets are forwarded to the exit domain using the address $G.E.d16$, and then forwarded to the entry domain $d6$ using the intra-domain routing table. Thus the packet reaches C.

When a packet is forwarded using a superdomain-level source route $\langle U_1, U_2, \ldots, U_n \rangle$, routers in U_i should only forward the packet to routers in U_i or U_{i+1} (otherwise, if a router in U_i forwards the packet to an intermediate domain d that is neither in U_i nor in U_{i+1}, then the resulting path may not be valid).

Hence, a router maintains for each visible sd-edge in its view an exit domain address and an entry domain-id (these attributes are in addition to cost, strong constraints and weak constraints). For the superdomain hierarchy of Figure 2, the views of two routers (one in domain $d1$ and one in domain $d16$) with these additions are shown in Figures 5 and 6.

[5] In fact, a domain-level source route may be needed, as we shall see later.

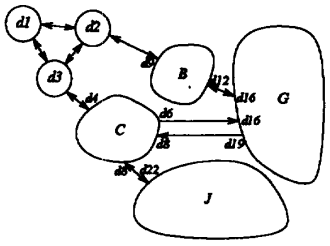

Figure 5: Full view of a router in $d1$.

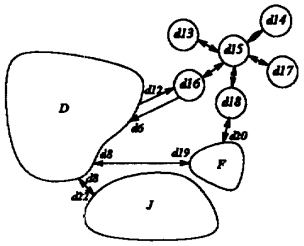

Figure 6: Full view of a router in $d16$.

4 Update Protocol using Raw-edges

Topology changes (e.g. node/link failures, cost changes) can affect the status of sd-edges. For example in Figure 5, if the d-edge $(d6, d16)$ failed, then the entry domain of sd-edge (C, G) would become invalid and its cost would change. Thus the attributes of sd-edges in the views of routers must be regularly updated. For this, the routers of each superdomain U must inform routers in U's parent of changes in sd-edges outgoing from U to every visible neighbor superdomain V. But it is not sufficient to report updates in terms of sd-edges (U, V), because of the differences in visibility between routers in U and routers outside U. For example in Figure 2, it is not sufficient for a router in $d6$ to report to $d16$ an sd-edge (D, G). This is because in the views of routers in $d16$, there are two edges $(D, d16)$ and (D, F) from D that participate in (D, G).

Hence routers in U need to identify the outgoing sd-edges with more resolution. Our solution is for the routers of U to report outgoing sd-edges (U, d) where d is a domain, along with the following attributes: (1) a cost, reflecting an "aggregate" cost of paths from any node in U to d over the edges participating in (U, d); (2) an exit domain address, which is the address of a domain e in U such that (e, d) is a d-edge; and (3) an entry domain address, which is the address of d. We refer to such sd-edges as *raw-edges*.

Receiver Aggregation

When a router outside U receives a raw-edge (U, d), it treats the information as pertaining to the sd-edge (U, V), where V is the superdomain visible to the router in the address of d. That is, the exit domain address, the entry domain id, and the cost of the raw-edge are used to update the corresponding attributes of the sd-edge (U, V) in the router's view. Note that the router can receive raw-edges $(U, d_1), (U, d_2), \ldots, (U, d_n)$ where the d_i's are different but the visible superdomains in their addresses are the same. In this case, the router uses the exit domain address and the entry domain id from one of these raw-edges, but aggregates the costs in these raw-edges into one cost. We refer to this as "receiver aggregation" of raw-edges. For example in Figure 2, a router in J would aggregate raw-edges from D with entry domain address $G.E.d16$ and $G.F.d19$ into the sd-edge (D, G).

Reporter Aggregation

We now describe how a router obtains for each of its ancestor superdomain U the outgoing raw-edges and their attributes.

If U is a domain, then the router constructs the raw-edges using its intra-domain routing table. Specifically, it obtains the entry domain address of each raw-edge (U, d) directly from the table (we assume that intra-domain routing table contains the addresses of the neighbor domains). It computes the cost of raw-edge (U, d) from the costs of the links in U and the costs of the links from U to d (these link costs are available in the routing table). The exit domain address is of course the address of U.

If U is a superdomain (and not a domain), then the router constructs the outgoing raw-edges from (1) the sd-edges between visible descendants of U, and (2) the raw-edges outgoing from the visible descendants of U to domains outside of U. This requires the router to maintain some additional information in its view, namely: raw-edges (V, d), where V is a visible superdomain and d is a domain outside V's parent (which is also a parent of the router). The router obtains this information from the raw-edges disseminated by the routers in V. For the example of Figure 2, the raw-edges maintained by routers in domain $d1$ and domain $d16$ are shown in Figures 7 and 8 respectively.

Let V range over the visible descendants of U. The router obtains the cost of a raw-edge (U, d) from the cost of the visible sd-edges in U and the cost of raw-edges (V, d). The router obtains the exit domain address of raw-edge (U, d) by arbitrarily choosing an exit

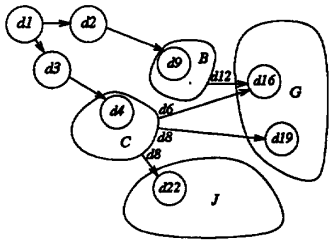

Figure 7: Raw-edges of a router in $d1$.

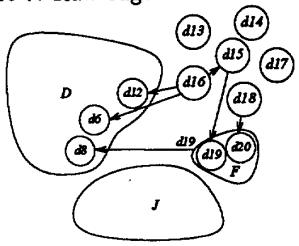

Figure 8: Raw-edges of a router in $d16$.

domain address of a raw-edge (V, d). We refer to this as "reporter aggregation" of raw-edges. For example, in Figure 2, when constructing raw-edges from D, a router in $d1$ can aggregate raw-edges $(B, d16)$ and $(C, d16)$ into the raw-edge $(D, d16)$ and use entry domain address $G.E.d16$.

5 Query Protocol

When a source wants a superdomain-level source route to a destination, it queries a router in its domain, which examines its view and searches for a valid path using the destination address[6]. We refer to this router as the *source router*. Even though the source router does not know the constraints of the individual domains that are to be crossed in each superdomain, it does know their strong and weak constraints. We refer to a superdomain whose strong constraints are satisfied as a *valid superdomain*. If a superdomain's weak constraints are satisfied but strong constraints are not satisfied, then there may be a valid path through this superdomain. We refer to such a superdomain as a *candidate superdomain*.

[6] We have assumed that the source has the destination's address. If that was not the case, it would first query the name servers to obtain the address for the destination. Querying the name servers can be done the same way it is done currently in the Internet. It requires nodes to have a set of fixed addresses to name servers. This is also sufficient in our case.

A path is valid if it involves only valid superdomains. A path involving a superdomain which is neither valid nor candidate cannot be valid. We refer to a path involving only valid and candidate superdomains as a ***candidate path.***

If there is a candidate path (and no valid path) to the destination in the view of the source router, then for each candidate superdomain U the source router queries any router of U to obtain an ***internal view*** of U. The internal view of U consists of the children of U and the sd-edges between them, and the raw-edges from these children to domains outside of U (and attributes of the children and edges). When the source router receives the internal view of U, it merges this with its view and searches for a valid path again. If there is still no valid path but there are candidate paths, the process is repeated. For example in Figure 2, the internal view of G is shown in Figure 9, and the resulting merged view at a router in superdomain $d1$ is shown in Figure 10. If $d16$ allows transit traffic from $d1$ while $d19$ does not, a valid path through $d16$ (since strong constraints of E are satisfied) can be discovered using this merged view.

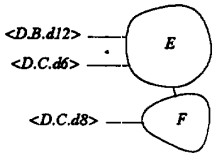

Figure 9: Internal view of G.

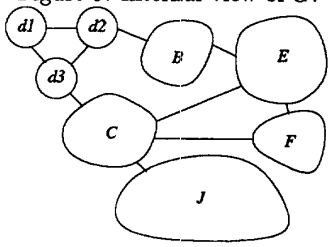

Figure 10: Merged view at $d1$.

When router x receives a request for the internal view of an ancestor superdomain U, it can construct the internal view of U as follows. Let superdomain V be the child of U that contains x. (1) x's view includes the siblings of V and the sd-edges between them. (2) x constructs the constraints of V from the constraints of the visible descendants of V. (3) x constructs the sd-edges outgoing from V into siblings of V by doing both reporter and receiver aggregation of raw-edges from the visible descendants of V to domains in siblings of V (these raw-edges are maintained since V is an ancestor of x). (4) x constructs the raw-edges outgoing from children of U into domains outside of U by doing reporter aggregation of the raw-edges leaving U.

When the source router receives the internal view of a superdomain U, it does the following. It removes U and its sd-edges (both into U and outgoing from U) from its view. It adds the children superdomains in the internal view of U to its view. It does receiver aggregation on the raw-edges from the children of U to superdomains outside of U. However, it treats a superdomain as visible during aggregation if the superdomain is in the current view of the source router. For each sd-edge (V, W) added to view, it also adds the sd-edge (W, V) to its view. The entry and exit domain addresses are copied by reversing those of (V, W). The cost of (W, V) is already in a raw-edge maintained by the router if this edge also leaves the parent of W. Otherwise, the cost of (W, U) is used as the cost of (W, V).

6 Topology Changes

Link cost changes and link/node failures and repairs correspond to cost changes, failures and repairs of sd-edges. Since a d-edge participates in many sd-edges, a topology change, such as a link failure, may cause a set of sd-edge failures.

Link/node failures can also partition a superdomain into cells, where a cell of a superdomain is defined to be a maximal subset of nodes of the superdomain that can reach each other without leaving the superdomain. In the same way, link/node repairs can merge cells into bigger cells. Superdomain partitions can occur at any level in the hierarchy.

To handle such topology changes, we use "cell-ids" instead of superdomain-ids. We identify a cell of a superdomain U by $U{:}x$, where x is the minimum total-id of the routers in the cell.

If a superdomain gets partitioned, its vertex in the views of routers should be split into as many pieces as there are cells. And when the cells merge, the corresponding vertices should be merged as well. Thus, the vertices in a view now stand for visible cells, sd-edges are now between superdomain cells, d-edges are now between domain cells, and raw-edges are now from superdomain cells to domain cells.

7 Conclusion

We have presented an inter-domain routing protocol based on superdomains, which always finds a valid path if one exists. Exit and entry domain attributes for sd-edges are maintained as a way to achieve superdomain-level source routing using intra-domain routing service. Routers exchange raw-edges to update sd-edges and their attributes.

Both strong and weak constraints are maintained for each visible superdomain. If the strong constraints of superdomains on a path are satisfied, then the path is valid. If the weak constraints of a superdomain are satisfied but the strong constraints are not, the source uses a query protocol to obtain a more detailed "internal" view of the superdomain, and searches again for a valid path.

Our protocol preserves the logarithmic space complexity of the classical area hierarchy approach [9], with the exception of merging views, where the storage requirement may be more than logarithmic in the worst case. Heuristics to choose candidate superdomains to query and to choose superdomains to delete from merged views should be investigated. Note that merged views need not be stored in fast expensive memory, since they are not used in packet forwarding.

The cost of merging views and forwarding packets over superdomain-level source routes can be amortized over packets by using connection-oriented routing (other approaches [6, 10, 5, 8] have similar cost).

In this paper, we have ignored how source policy constraints of a domain are adapted to the superdomain hierarchy. Having source policy constraints only at the domain level is not sufficient, since source routers do not necessarily know which domains would be crossed in each superdomain. Hence, source policy constraints should also be stated for superdomains. This can also be done using strong and weak bounding techniques[7]. We have ignored source policy constraints because source routers have more flexibility expressing their own constraints in any level of detail (for example, a source router, for a specific superdomain, can always obtain the internal view, and apply source policy constraints to children superdomains).

[7] E.g. if the strong (weak) policy constraints is used, source has a policy constraint for a superdomain if it has that policy for some (all) children superdomains.

References

[1] C. Alaettinoğlu and A. U. Shankar. Hierarchical Inter-Domain Routing Protocol with On-Demand ToS and Poicy Resolution. Technical Report UMIACS-TR-93-18, CS-TR-3038, Department of Computer Science, University of Maryland, College Park, March 1993.

[2] C. Alaettinoğlu and A. U. Shankar. Viewserver Hierarchy: A Scalable and Adaptive Inter-Domain Routing Protocol. Technical Report UMIACS-TR-93-13, CS-TR-3033, Department of Computer Science, University of Maryland, College Park, February 1993.

[3] A. Bar-Noy and M. Gopal. Topology Distribution Cost vs. Efficient Routing in Large Networks. In Proc. *ACM SIGCOMM '90*, pages 242–252, Philadelphia, Pennsylvania, September 1990.

[4] L. Breslau and D. Estrin. Design of Inter-Administrative Domain Routing Protocols. In Proc. *ACM SIGCOMM '90*, pages 231–241, Philadelphia, Pennsylvania, September 1990.

[5] J. N. Chiappa. A New IP Routing and Addressing Architecture. Big-Internet mailing list., 1992. Available by anonymous ftp from `munnari.oz.au:big-internet/list-archive`.

[6] D.D. Clark. Policy routing in Internet protocols. Request for Comment RFC-1102, Network Information Center, May 1989.

[7] D. Estrin. Policy requirements for inter Administrative Domain routing. Request for Comment RFC-1125, Network Information Center, November 1989.

[8] D. Estrin, Y. Rekhter, and S. Hotz. Scalable Inter-Domain Routing Architecture. In Proc. *ACM SIGCOMM '92*, pages 40–52, Baltimore, Maryland, August 1992.

[9] L. Kleinrock and F. Kamoun. Hierarchical Routing for Large Networks. *Computer Networks and ISDN Systems*, (1):155–174, 1977.

[10] M. Lepp and M. Steenstrup. An Architecture for Inter-Domain Policy Routing. Internet Draft. Available from the authors., June 1992.

[11] R. Perlman. Hierarchical Networks and Subnetwork Partition Problem. *Computer Networks and ISDN Systems*, 9:297–303, 1985.

[12] Y. Rekhter. Inter-Domain Routing Protocol (IDRP). Available from the author., 1992. T.J. Watson Research Center, IBM Corp.

[13] P. F. Tsuchiya. The Landmark Hierarchy:A New Hierarchy For Routing In Very Large Networks. In Proc. *ACM SIGCOMM '88*, August 1988.

[14] P. F. Tsuchiya. Efficient and Robust Policy Routing Using Multiple Hierarchical Addresses. In Proc. *ACM SIGCOMM '91*, pages 53–65, Zurich, Switzerland, September 1991.