

QoS Routing Across Multiple Autonomous Systems Using the Path Computation Element Architecture

Geza Geleji[†] and Harry G. Perros

ggeleji@ncsu.edu, hp@csc.ncsu.edu

North Carolina State University, Department of Computer Science

Raleigh, NC 27695–8206

Abstract

We examine various algorithms for calculating QoS paths spanning multiple autonomous systems (ASs) using the Path Computation Element (PCE) architecture. The problem is divided into two parts. We first calculate an AS-path, then the node-by-node path. Using extensive simulation, we compared various AS-path calculation algorithms based on BGP and various AS aggregation procedures, such as mesh, star and nodal aggregation. For node-to-node path calculation, we employed the Per-Domain Backward algorithm and the Per-Domain Backward Tree algorithm (also known as Backward Recursive PCE-Based Computation). Results point to the fact that complex AS-path calculation algorithms do not perform significantly better than BGP. However, if the service quality provided by ASs varies greatly, either in time or space, then we expect a QoS-aware AS-path computation algorithm, e.g. static nodal aggregation, to outperform BGP. Although the Per-Domain Backward Tree algorithm generally performs better than the Per-Domain Backward algorithm, using a persistent variant of the latter makes it outperform the Per-Domain Backward Tree algorithm. The cost is a negligible increase in computational complexity and a slightly increased connection setup delay.

I. INTRODUCTION

Path Computation Elements (PCEs) [1] are the core constituents of a novel routing architecture in which the routing and forwarding planes are separate (similarly to the Routing Control Platform proposed by Feamster et al. [2]). Routes are computed by PCEs (possibly through complex, constraint-based routing algorithms) and then applied to the forwarding plane, which is driven by a connection-oriented control plane like MPLS. This architecture benefits from the more specialized functionality of said components; as the only responsibility of PCEs is route computation, we expect them to be able to do it more efficiently, i.e., to do complex computation for the purposes of QoS-aware routing. We also expect PCEs to be able to perform admission control; Shenker [3] argues that

[†] Corresponding author

this is necessary for applications with hard real-time QoS requirements. The user perceived quality of a voice telephony application declines sharply when either the end-to-end packet delay (referred to as One-Way Delay in Amante et al. [4]; see also IP Packet Transfer Delay in ITU-T Y.1540 [5]) or the range of the IP Packet Delay Variation ([4], [5]; see [5, Section 6.2.4.2] for a quantile-based range characterization) goes above a well-defined threshold. Streaming video typically requires a much larger bandwidth than voice telephony, but can tolerate higher delays, delay variations and packet loss rates. Many implementations of the widely used Transmission Control Protocol (TCP) are able to tolerate long delays and a large delay variations, but are rather sensitive to packet losses. Clearly, different applications have different QoS requirements in terms of, to cite some of the most important QoS metrics, bandwidth, end-to-end delay, packet delay variation, packet loss, and availability. In the present-day Internet, the most widely used routing protocols are BGP and OSPF. Neither of them was designed for QoS routing and experience shows that using them for this purpose yields poor results.

Finding an optimal path constrained by multiple end-to-end QoS metric requirements is a problem whose best solutions, in general, scale very poorly to large networks. In fact, many of the basic related problems have been shown to be NP-complete. For example, the goal of the Multi-Constrained Optimal Path problem [8] is to find a minimum-cost path with respect to a single additive cost metric while simultaneously conforming to constraints posed on the path through other additive cost metrics. The NP completeness of several other similar problems involving additive, multiplicative, convex and concave metrics and greater-than-equal and less-than-equal constraints is shown in [9]. For this reason, numerous attempts have been made to develop scalable routing algorithms. A considerable proportion of efficient routing algorithms rely on a hierarchical network representation (e.g. the PNNI routing protocol in ATM [6]), in which a large network is partitioned into smaller, disjoint subnetworks (we will refer to this instance of partitioning as the “top level” of the hierarchy); subnetworks may also be further partitioned into sub-subnetworks, and so on, until the partitions are small enough to make any further division unreasonable (we will call this the “bottom level” of the hierarchy). Each entity in this hierarchy is only aware of its direct constituents; i.e., the top level network is only aware of its subnetworks, and a subnetwork is only aware of its own direct sub-subnetworks. A possible way to find an end-to-end route in this hierarchical setup would be to traverse the hierarchy level by level, from top to bottom. At the top level, identify a path of subnetworks between the subnetworks containing the two endpoints, then pass this subnetwork path to the level below, and refine it into a path of sub-subnetworks. When the process ends on the bottom level of the hierarchy, the path will have been refined into a node-by-node path.

A particularly important aspect is how the partitioning of a network is done. In fact, the partitioning procedure may hide details about the partitions themselves, i.e., subnetworks may present a simplified image of themselves to their parent network. The literature refers to this process of simplification as “network aggregation”. Without aggregation, the top level of the hierarchy would constitute an exact view of the whole network. This is not desirable when trying to achieve scalability, since finding the path of subnetworks would already require computing an exact end-to-end, node-by-node path. With appropriate network aggregation, however, no component has a complete view of the network. This improves scalability at the cost of complicating and reducing the effectiveness of routing

itself. An example of aggregation would be to represent a subnetwork by its boundary nodes only (i.e., nodes that are directly connected to other subnetworks), interconnected by direct links calculated by taking into account every possible path within the subnetwork. We will refer to this aggregation scheme as “full-mesh aggregation”. The more information the aggregation retains, the more precise and less scalable the routing will be. Conversely, the more it discards, the more scalable and less accurate the routing will become. Several possible aggregation schemes will be discussed and evaluated later in this paper.

Matters are further complicated when the aggregation procedure has to deal with multiple QoS metrics. As an example, let us consider a full-mesh aggregation of a network in which there are two possible paths between the boundary nodes A and B . Let us assume that the first path has a mean end-to-end delay of 600 msec and there is 1 Gbps of bandwidth available on it, while the same measures are 20 msec and 1 Mbps for the second one. The question is which of the two paths we will use to represent the link between A and B in the full mesh aggregation. If we would like to forward a VoIP call between the endpoints, then the second path is clearly better. However, for streaming video, the first one is more suitable. The problem gets more complicated if we include other metrics, such as jitter and packet loss. The problem is how do we select a path based on multiple different metrics and how do we summarize link bandwidth when links have different capacity. Multi-attribute routing has been addressed; see [8] and [10]. However, the problem of aggregating bandwidth along a path of different capacity links remains open.

Without loss of generality, we will limit the scope of our paper to a simplified network architecture with two hierarchy levels: the Autonomous System level and the node level. That is, our network consists of autonomous systems interconnected through their boundary nodes to form an arbitrary topology, and each autonomous system consists of nodes, also arbitrarily interconnected. The hierarchical route computation process consists of two parts: first, find an AS path, which is a sequence of ASs where the first AS contains the source node of the connection and the last AS contains the sink node; second, refine the AS path into an end-to-end sequence of specific nodes. Note that the routing procedure used in the Internet employs a similar two-level structure: the lower layer is served by OSPF, while the upper layer by BGP. We have implemented several algorithms for both parts in a simulator and compared their performance. We note that our results can be extended to networks with more than just two hierarchy levels, but this case was not considered here.

The structure of our paper is as follows. In the next section, we describe the network simulator, and in Section III, we describe the path computation algorithms that we implemented. Section IV discusses the simulation results and the conclusions are given in Section V.

II. NETWORK MODEL AND SIMULATION ENVIRONMENT

We have developed a network simulator to simulate the control plane of a connection-oriented IP network in which routing is performed by PCEs. In the core of the simulator lies a graph of the network, where the graph nodes correspond to the network nodes, pairs of which may be connected by unidirectional links — in which case they are represented by unidirectional edges in the graph. Nodes are assigned a pair of spherical coordinates on a

sphere whose radius is approximately equal to the mean radius of the Earth. Assuming that links lie along great circles, one can determine their approximate physical length, and consequently, their propagation delay. Edges are also assigned capacity values, which represent their bandwidth.

We assume that the underlying IP network is MPLS enabled; that is, in order for two routers to communicate, a connection, known as a label switched path (LSP), has to be first established using RSVP-TE. This connection is up for the entire duration of the communication and then it is torn down [7]. In the simulation model we only consider a traffic class which is delay sensitive, such as the DiffServ Expedited Forwarding (EF) class. Setup requests for such a traffic class include the peak rate, which is used to allocate bandwidth on the links along the path. Due to peak rate bandwidth allocation, packets belonging to these connections are not delayed at the output buffers of the routers. Thus, the end-to-end delay is just the propagation delay (assuming zero delay at each router).

In our simulation, the QoS metrics requested by such connections are the end-to-end delay and the peak rate bandwidth (other metrics, such as jitter and packet loss may be taken into account using composite metric; this case is beyond the scope of this paper). We note that in the simulation experiments, we only take into account the requested bandwidth but not the requested end-to-end delay. That is, the objective is to calculate a path whose links satisfy the minimum bandwidth requirement, and, which has the smallest end-to-end propagation delay.

In addition to the computational aspects of routing, PCEs also perform resource management. Since these tasks may be quite complex, the acting scope of a PCE is limited to a single AS. In our simulation model, each AS is served by exactly one PCE embedded in a network node within the AS. Each network node is statically configured to direct all connection establishment requests to that PCE. It is possible to have multiple PCEs per AS, but this case was not considered here.

Resource allocations are handled by a sub-unit within the PCEs called the Traffic Engineering Database (TED). A TED describes the state of all links within an AS; for each link, it lists all connections that have bandwidth assigned to them on the link as well as the amount of bandwidth assigned. When a new connection is admitted, new entries are added to the TED for each link (within the domain) traversed by the connection. These entries identify the link itself, the connection and the requested bandwidth. If there is not enough available capacity on any of the traversed links, the allocation operation fails and is normally rolled back. When a connection is terminated, all entries belonging to it are removed. At any instant, the sum of the bandwidths allocated by the entries belonging to a given link equals to the total bandwidth used on the link; subtracting this value from the capacity of the link gives the available bandwidth, which is checked every time a new allocation is requested on the link.

Our program simulates the route computation process including control message exchanges between PCEs, as well as bandwidth reservations and releases. In a typical connection setup, a customer notifies a nearby PCE through a Path Computation Request [12] message. The PCE then starts the process of path computation, which normally involves several message exchanges between PCEs belonging to different domains. During this time, an effort is made to identify an end-to-end path for the connection, and to reserve the requested capacity on all links traversed by it. When the reservation is done, a Path Computation Reply message indicating the success is sent back to the originator. The effort may fail if a path with the requested capacity available does not exist between

the endpoints, in which case the Path Computation Reply indicates failure. Upon the termination of the connection, which occurs in the simulation when the pre-determined holding time has elapsed after the successful establishment of the connection, all allocated resources are freed. In reality, this resource release may also require inter-operation between the PCEs serving different ASs along the path of the connection. However, for the sake of simplicity, we assume that as soon as the connection is terminated, the PCEs remove all entries belonging to the connection from their respective TEDs in zero time.

We note that the simulation model only simulates the PCE control plane. Once a path has been calculated and signalled to the source router, the router launches an RSVP-TE path message towards the destination with the path indicated in its Explicit Route Object. Data flows once the path has been established. Neither the establishment or tearing down of the actual MPLS connection, nor the transmission of data on the connection is simulated. Instead, the bandwidth allocated to the links along the connection is held for an exponentially distributed amount of time, which reflects the duration of the connection.

A. Test Network Topologies

The simulator treats all links as unidirectional. However, in our test networks, connections are symmetric: if there is a link from node A to node B , then there is also a link from B to A . Each domain has exactly one PCE. We have simulated two network topologies, referred to as ‘A’ and ‘B’. The parameters of the topologies are presented in Table I. Topology ‘A’ has fewer ASs with a larger number of nodes per AS, than topology ‘B’. There are more intra-AS links and fewer inter-AS links in topology ‘A’ than in ‘B’. As a result, there are more paths available within an AS in topology ‘A’, whereas there are more AS paths in topology ‘B’. We have conducted experiments on several other topologies as well, but we have found these two topologies produce results that are diverse enough for the purpose of comparing various AS-path computation algorithms.

B. Traffic Samples

We have created different sets of traffic samples for each test network. Each sample consists of a sequence of connection setup requests. A connection setup request is defined by the connection inter-arrival time measured from the previous request, the source and sink nodes, the required capacity and the requested holding time. Source and sink nodes are picked by first picking a domain using the uniform distribution, then picking a node within the domain, again, following the uniform distribution. If the same node is picked as both source and sink, then a new sink node is chosen within the same domain. The capacity value also follows a uniform distribution on the set of integers in $[1, 10]$. The holding time is distributed exponentially with a mean of 50 time units. The arrival of setup requests follows a Poisson process with a given rate constant throughout a sample, which was varied from 2^0 to 2^3 , in increments of $2^{0.25}$. For each test topology and arrival rate, we created 4 independently randomized traffic samples with the same parameters, and the confidence intervals were obtained using Student’s t -statistic. Since we have 13 different arrival rates, the total number of traffic samples was 4×13 for each test network.

TABLE I: Network Topology Parameters

	Topology 'A'	Topology 'B'
No. of ASs	13	40
No. of nodes	151	
Mean no. of nodes per AS	11.62	3.78
No. of nodes in smallest AS	6	3
No. of nodes in largest AS	18	6
No. of links	458	
No. of intra-AS links	366	242
No. of inter-AS links	92	216
Mean link length	158.87 km	197.54 km
Mean intra-AS link length	128.18 km	117.39 km
Mean inter-AS link length	280.96 km	287.4 km
Link capacity (uniform)	40	
Mean in / out degree	3.03	

III. THE PATH COMPUTATION PROCESS

a

The two-level hierarchical structure of the network prompts the path computation to be structured in a similar way. In the first phase of the process, which we will refer to as the “AS path computation”, we find a path of ASs between the ASs containing the endpoints. We will refer to the endpoints as “source” and “sink”, and the ASs containing them as “source AS” and “sink AS”, respectively. The AS-level path is a sequence of ASs from the source AS to the sink AS within which a unidirectional end-to-end node-level path between the endpoints will be established. The second phase is called “node-by-node path computation”. It calculates within a given AS-level path obtained in the first phase a node-by-node path from the source to the sink.

A. Phase 1: Computing the AS-level path

All of the AS path computation algorithms we implemented rely on the following scheme. First, they construct an aggregate topology of the network, then they find the shortest path between the aggregate nodes representing the source and the sink nodes of the requested connection, and finally, they identify the sequence of ASs traversed by this path and return it as the result of the computation. The aggregation schemes considered in this paper are described below. We note that some of the schemes cannot be realistically implemented. They were introduced merely for comparison purposes.

1) *The Border Gateway Protocol:* One of our AS path computation procedures was designed to resemble the Border Gateway Protocol. Although it is by no means an exact implementation of the real-life routing algorithm, its behavior approximates that of BGP. Specifically, given two endpoints, it finds the AS path connecting the two which consists of the smallest possible number of AS hops (i.e., inter-AS links). The algorithm was implemented as follows. First, an aggregate graph model is constructed in which every AS is represented as a single node. An edge is added between two nodes if there is a link connecting any node in the respective AS of the first node to any node belonging to the respective AS of the second node. Every edge is assigned a unit cost. It is easy to see that for every pair of endpoints, this algorithm will return the AS path with the least AS hops, irrespective of whether a node-by-node path with enough available capacity actually exists within the AS path between the endpoints. The aggregate graph model does not change as resources are allocated and released, since it does not take resource allocations into account. For this reason, the aggregation procedure is carried out only once, at the beginning of the simulation, on an empty network.

2) *Static Full Mesh Aggregation:* The full mesh aggregation algorithm substitutes domains with their ingress and egress nodes. An edge is drawn in the aggregate graph from an ingress node to an egress node if there is a path from the ingress node to the egress node in the AS. The cost of the link is set equal to the cost of the network path, the cost being the propagation delay. This algorithm provides an accurate depiction of the network for the purposes of AS path computation at the cost of poor scalability; note that a link is added to the aggregate graph for each possible path from an ingress node to an egress node. For an AS with u ingress nodes and v egress nodes, $\Theta(uv)$ links would be added to the aggregate graph. This algorithm is called “static” because the aggregation is performed only once at the beginning of the simulation, on an empty network. That is, connections in the aggregate graph are determined based on the physical connectivity of the network, and resource availability is not accounted for. The AS path is computed by running Dijkstra’s shortest path algorithm on the aggregate graph and enumerating the ASs traversed. The AS path returned by this algorithm is independent of the state of the network and it may or may not contain an actual path with sufficient available resources to route a particular connection. For this reason, the aggregation procedure is carried out only once, at the beginning of the simulation, on an empty network.

3) *Real-Time Full Mesh Aggregation:* This is similar to the algorithm above, but it performs the aggregation for every request, ignoring the links in the AS that do not have sufficient capacity to satisfy it. Thus, the AS path is deduced from completely up-to-date information and is guaranteed to contain a node-by-node route with sufficient available resources for the connection. In practice, this algorithm is not useful due to its high computational cost, but because of its good performance, it is used for comparison purposes in our paper.

4) *Star Aggregation:* Star aggregation tries to improve scalability, as compared to the full mesh approach, by creating a “central node” in the aggregate view of each domain, then adding edges from each ingress node to the central node, as well as from the central node to each egress node. Instead of using $\Theta(uv)$ links as in the full mesh scheme, this algorithm uses only $\Theta(u + v)$. Edge costs are determined using the least-squares approximation of the shortest paths connecting the ingress and egress nodes, that is, edge costs are chosen so that the sum of squares of the differences between the costs of the shortest paths from the ingress to the egress nodes in the real network

and their counterparts in the star topology is minimized.

The static version of this aggregation scheme carries out the network aggregation only once, at the beginning of the simulation, based on the physical connectivity of the network. Resource availability is not considered, therefore the computed AS path is independent of the network state. We have also implemented the real-time version of this algorithm, which, as opposed to its static counterpart, ignores links without sufficient capacity and updates the aggregate graph every time a connection request is received. However, owing to the inaccurate representation created by the least-squares fit, finding successfully an AS path in the aggregate graph does not guarantee the existence of a node-by-node path in the real network. In addition, experience shows that the static and real-time variations give almost the exact same results, therefore we will only present results for the static version.

5) *Nodal Aggregation*: Nodal aggregation is a variation of the star aggregation. Instead of one central node, two central nodes are created in the aggregate graph for each domain, namely, an ingress central node and an egress central node. The edge costs are determined very differently. For each ingress node, an edge from the ingress node to the ingress central node is added to the aggregate graph, and for each egress node an edge is added from the egress central node to each egress node. However, unlike in the star aggregation, the costs of these links are all set to zero. The only edge with a nonzero cost is the one connecting the ingress central node to the egress central node. Its cost is known as the “diameter” of the domain and it is the average cost of a path from any ingress node to any egress node in the domain.

We have implemented both static and real-time variations of the algorithm. In the static version, aggregation is carried out only once, at the beginning of the simulation, on an empty network. In the real-time version, aggregation is done every time a connection request is received. However, experience shows that re-aggregating the network before every connection does not change the diameter significantly and therefore, the two algorithm give almost the exact same results. For this reason, we will only present results for the static version.

B. Phase 2: Computing The Node-by-Node Path

Node-by-node path computation typically, with the exception of the Instantaneous Flat algorithm, requires an AS path as input and provides a node-by-node path as the output. The ultimate goal of the process is to find the least-cost node-by-node path within the AS-path, as quickly as possible, where the cost is always the propagation delay.

1) *The Instantaneous Flat Algorithm*: We assume that the entire network topology is known, and the complete network state is readily accessible to the PCE performing the computation. A graph is created from the topology and all links that do not have sufficient available capacity to accommodate the connection are excluded from it. An exact node-by-node shortest path is identified by running Dijkstra’s algorithm on the graph. (Recall that in this topology, the edge costs represent the propagation delays.) The computation is carried out by a single PCE, no message exchanges are involved, and the process is assumed to take zero time. Clearly, this algorithm gives the lowest cost path and is used for comparison purposes when evaluating the other algorithms. If in any given state of the network, a node-by-node path exists between the endpoints, then this algorithm is guaranteed to find it.

2) *The Per-Domain Backward Algorithm*: A description of this simple algorithm can be found in Geleji et al. [13]. In this algorithm, the path is established backwards from the sink to the source by finding the best path within each AS. A PCE executing this procedure forwards each incoming path computation request to the PCE in the sink AS – we will refer to this PCE as the “end PCE”. Since PCEs are aware of the interdomain links to all adjacent ASs, the end PCE finds the cheapest path from the destination node to the penultimate AS and allocates the requested amount of bandwidth along the newly computed path segment which is terminated at an egress node, referred to as A , of the penultimate AS. When the path segment has been set up, the end PCE forwards the request to a PCE in the penultimate AS, whose task is to find the cheapest path from an egress node of its preceding AS, call it B , to A . When this is done, resources are allocated along the new path segment and the two path segments are merged to form a single segment, from B to the sink node of the path. This procedure is repeated until the source AS is reached, where a PCE finds a path segment from the requested connection source to its egress node, which marks the beginning of the segment to the destination node. If the establishment of the path fails for any reason, all associated resources in all domains are freed immediately. This algorithm finds the shortest path within each AS given that the egress node has been fixed. In view of this, it is not guaranteed to find the shortest path between the source and the destination nodes; however, its relatively low computational complexity and communicational overhead make it a promising candidate.

During the execution of the algorithm by the relevant PCEs, path computation is done with the exclusion of all links that do not have sufficient capacity to accommodate the connection. As a result, it is possible that a PCE may not be able to find a path through its AS, and therefore the establishment of the connection will fail. All resources allocated to the connection are freed immediately, and a message stating the failure of the process is propagated towards the PCE in the source AS, which, in turn, will forward the information to the Path Computation Client (PCC) originally requesting the connection. However, we have also implemented a “persistent” version of the algorithm. In this variation, the PCE does not immediately give up upon failure to find a path through its AS. Instead, it waits for a predefined amount of time and tries again, while all the resources reserved for the connection so far remain allocated. The total number of retries allowed for a single connection is set to a predefined value, and failure is only stated after this number has been exceeded. The persistent version has the potential to provide a lower blocking probability at the cost of increased connection set-up time. We note that we have not considered the case of crankback, whereby a different AS path is computed starting either from the sink AS or from the last AS through which a path was successfully computed. Our approach strives to achieve a similar effect at a lower computational cost.

3) *The Per-Domain Backward Tree Algorithm*: This algorithm was proposed by the PCE working group in RFC 5441 [11] and it is referred to as the Backward Recursive PCE-Based Computation (BRPC) algorithm. It builds a tree of shortest paths leading to the sink node, AS by AS. The construction of the tree starts at the sink AS and proceeds towards the source AS. Each AS encountered along the way contributes its own part. It is similar to the Per-Domain Backward algorithm, but instead of propagating a sub-optimal path from the destination node towards the source, a tree of all available paths is forwarded. As the tree travels along the AS path in reverse direction, a

PCE in each AS extends it by adding a set of paths from the egress nodes of the preceding AS to the leaves of the tree, keeping in the tree only the shortest paths between the leaves and the root. The time it takes for this algorithm to execute is similar to that of the Per-Domain Backward algorithm. This procedure causes more communication overhead as trees have to be propagated instead of path segments, however, it compensates for this shortcoming by finding paths that are closer to optimality. Obviously, this algorithm does not necessarily find the optimal path either because it is constrained by the pre-determined AS path which may not include the optimal end-to-end path. However, if the predetermined AS path contains the optimal end-to-end path, then this algorithm is guaranteed to find it. Also, the algorithm will not be able to find an existing path outside the boundaries of the specified AS path (e.g. if the only node-by-node path goes through the AS sequence 1-2-3-4-3-4-5, but the AS path given is 1-2-3-4-5, then the algorithm will fail). Note that in our implementation, the PCE responsible for path selection will always select the minimum cost path from the tree. If that path has become unavailable in the meantime due to resource allocation by other connections, the path computation process will fail and no other paths from the tree will be considered, even though there may be one which is still available. It should be noted that a minor difference between our implementation and the RFC 5441 proposal is that the connection setup request is always forwarded from the PCE in the source AS to the PCE in the sink AS along an optimal path. In a realistic scenario, this is generally impossible. The original BRPC algorithm forwards the request from one PCE to a PCE in the next AS, which infers a sub-optimal end-to-end path and additional processing delays.

We note that a comparison of the performance of these above algorithms along with some other algorithms for the case where the AS path is known in advance is given in Geleji et al. [13].

IV. SIMULATION RESULTS

In this section, we compare the AS path computation algorithms described in the previous section using our simulator. Each AS path computation algorithm is evaluated using the following three node-by-node path computation procedures: the Per-Domain Backward algorithm with no retries, the persistent variation of the Per-Domain Backward algorithm with a maximum of 10 retries, 10 msec apart, and the Per-Domain Backward Tree algorithm. The following performance measures are estimated as a function of the arrival rate of requests: the blocking probability (approximated by the number of blocked connections divided by the total number of offered connections in the sample), the mean connection setup delay (how long does it take, on the average, to successfully set up a connection from submitting the Path Computation Request message to receiving a Path Computation Reply message stating success), and the mean end-to-end delay (the propagation delay of the computed path between the endpoints of the connection). The results obtained are presented in a series of graphs. The confidence intervals have been calculated by running the four independent randomized traffic samples created for each topology and arrival rate, which form four independent replications for each data point, and subsequently using Student's t -statistic. The resulting interval estimates are very small and they are not discernible in the graphs. In view of this, they have been omitted from the presentation of the results.

In Figures 1, 2, and 3, we present results for the nonpersistent Per-Domain Backward Algorithm, the persistent

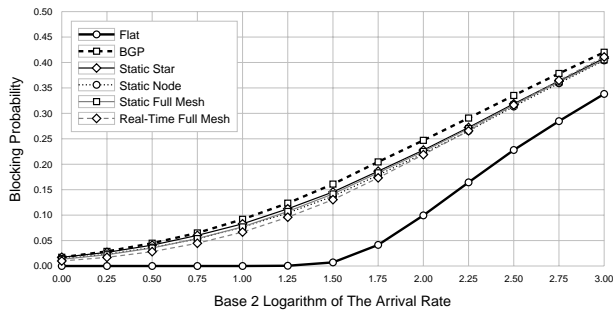
Per-Domain Backward Algorithm and the Per-Domain Backward Tree Algorithm, respectively. Figures 4 and 5 show comparisons of the node-by-node path computation algorithms investigated in our paper when used in conjunction with the static nodal aggregation algorithm and the real-time full mesh aggregation algorithm, respectively.

A. *The Instantaneous Flat Algorithm*

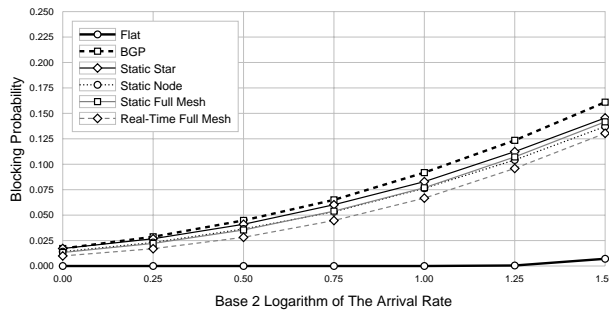
This reference algorithm exhibits a behavior that is distinct from the that of the other algorithms and is noteworthy for the better understanding of the explanations that will appear later in this section. Figures 1f, 1h, 2f, 2h, 3f, and 3h show that as the arrival rate increases, the mean end-to-end connection delay (and, accordingly, the mean end-to-end path length) increases. The instantaneous flat algorithm always finds the shortest path between two endpoints if one exists. If the network load is high, this path will typically be longer than if the load is low. However, unlike, the instantaneous flat algorithm, the other algorithms are constrained by AS paths: the node-by-node path they may return always lie within the boundaries of a given AS path and this limits its maximum length. If the network load is high and the given AS path does not contain a node-by-node path, the algorithm, instead of stepping outside the boundaries of the AS path, fails. Connection requests with shorter AS paths will be more likely to succeed, and for this reason, the average node-by-node path length will decrease.

B. *Full Mesh Aggregation*

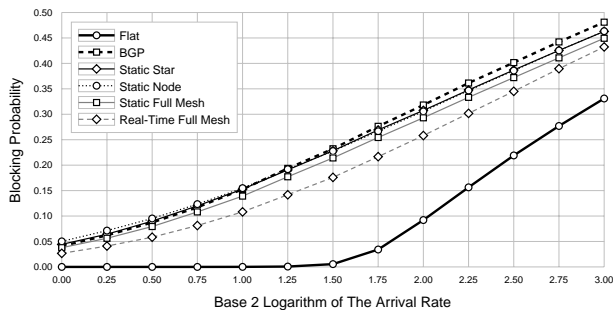
The real-time variation of the algorithm clearly gives the best performance among the aggregation-based algorithms when used in conjunction either with the nonpersistent version of the Per-Domain Backward algorithm (see Figures 1a, 1b, 1c, and 1d) or the Per-Domain Backward Tree algorithm (see Figures 3a, 3b, 3c, and 3d). This is not the case with the persistent Per-Domain Backward algorithm (see Figures 2a, 2b, 2c, and 2d), where, curiously, if the arrival rate exceeds a certain threshold, the blocking probability rises above that of all other algorithms. This may be explained as follows. Since the AS path is computed at the moment the setup request arrives, it is guaranteed that a node-by-node path between the end nodes exists. However, the Per-Domain Backward Algorithm takes up time to traverse all the ASs and there is no guarantee that the links used in computing the AS path will have the necessary capacity due to activity by other requests. However, what causes the drastic increase of the blocking probability as compared to the non-persistent variant for the Real-Time Full Mesh Aggregation, is the fact that the delay the connection setup may suffer at each AS prolongs the setup time, thus further reducing the probability that upstream links are still available. Note that Figures 1f, 1h, 2f, and 2h show that the mean end-to-end connection delay, and in parallel, the mean end-to-end connection length, is higher for the persistent variant. The cause is that the persistent variant keeps looking for the node-by-node path over a longer period of time (as opposed to the nonpersistent variant's instantaneous attempt) and is more likely to succeed, but this comes at a price: the mean path lengths increase as a consequence of the higher network load. Therefore, if the network load is relatively high, a connection request that has been assigned a long AS path is much more likely to fail than another which has been assigned a short AS path. Since the Real-Time Full Mesh Aggregation is much more likely to find connectivity between a given pair of endpoints than any other of the aggregation-based algorithms, the average length of the AS



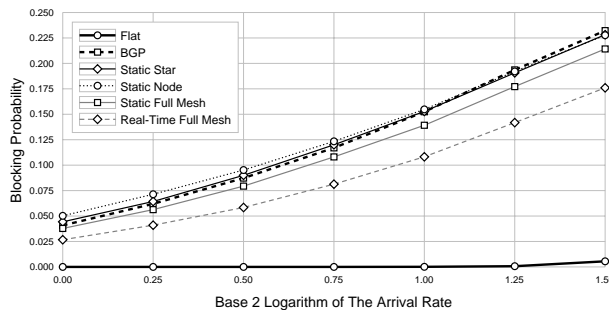
(a) Blocking probability vs. arrival rate, 'A' topology



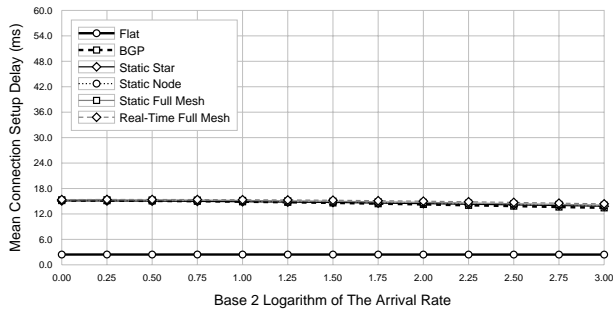
(b) Blocking probability vs. arrival rate, 'A' topology, magnified view



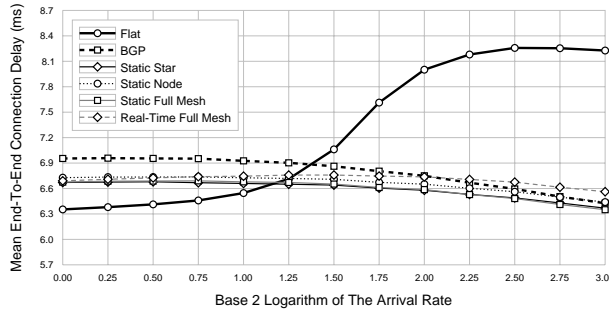
(c) Blocking probability vs. arrival rate, 'B' topology



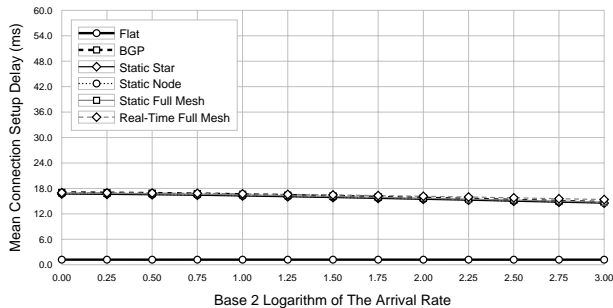
(d) Blocking probability vs. arrival rate, 'B' topology, magnified view



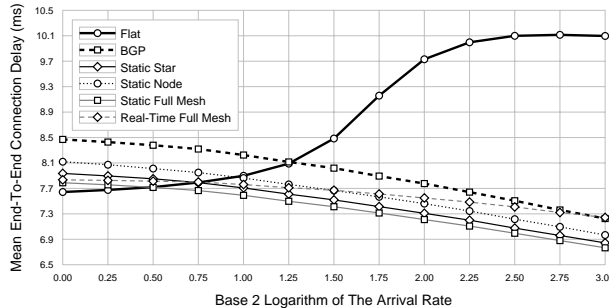
(e) Connection setup delay vs. arrival rate, 'A' topology



(f) End-to-end connection delay vs. arrival rate, 'A' topology

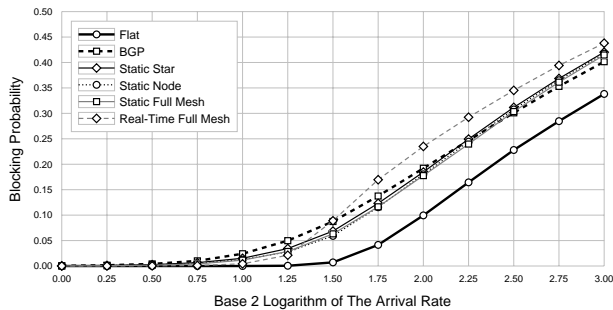


(g) Connection setup delay vs. arrival rate, 'B' topology

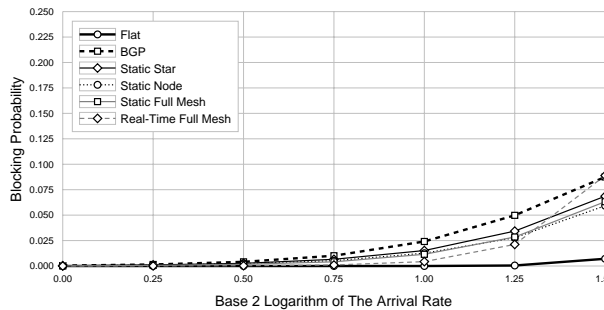


(h) End-to-end connection delay vs. arrival rate, 'B' topology

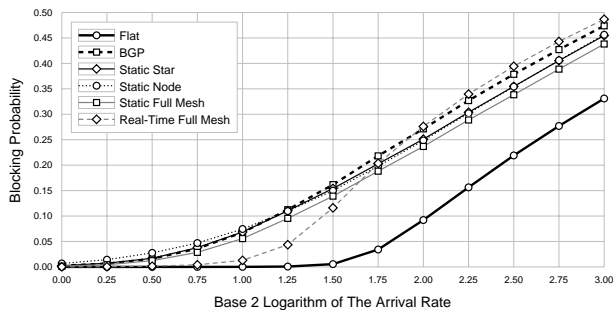
Fig. 1: Results for the nonpersistent Per-Domain Backward Algorithm



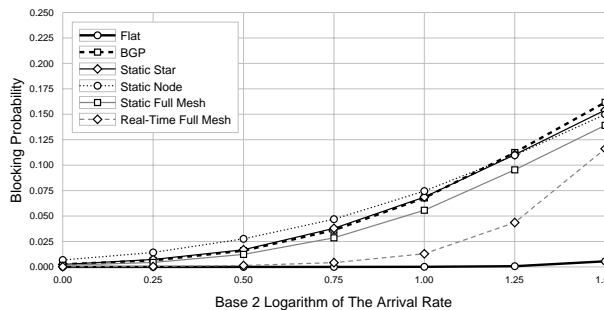
(a) Blocking probability vs. arrival rate, 'A' topology



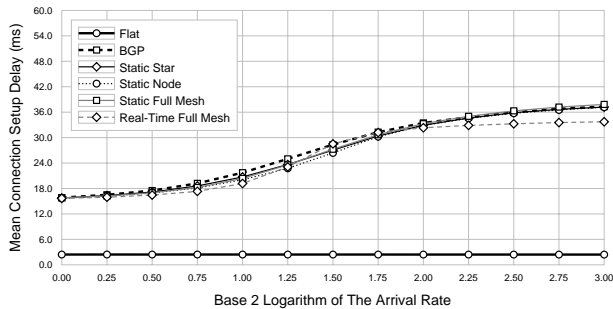
(b) Blocking probability vs. arrival rate, 'A' topology, magnified view



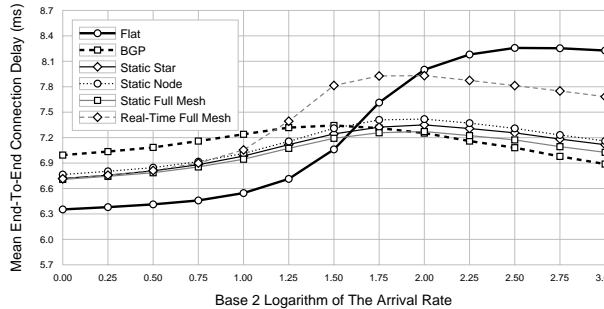
(c) Blocking probability vs. arrival rate, 'B' topology



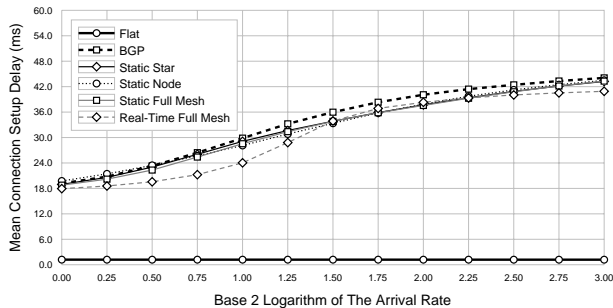
(d) Blocking probability vs. arrival rate, 'B' topology, magnified view



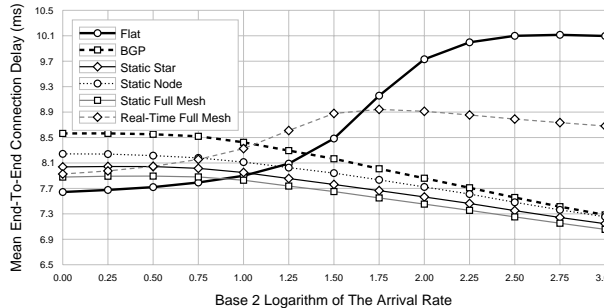
(e) Connection setup delay vs. arrival rate, 'A' topology



(f) End-to-end connection delay vs. arrival rate, 'A' topology

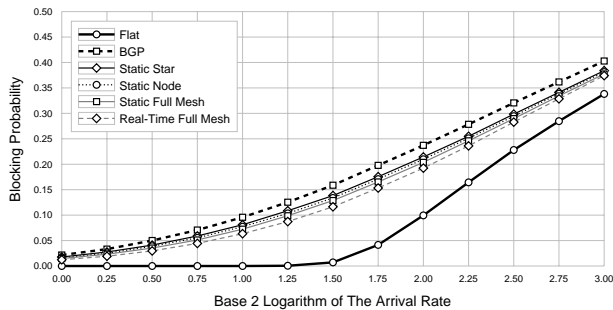


(g) Connection setup delay vs. arrival rate, 'B' topology

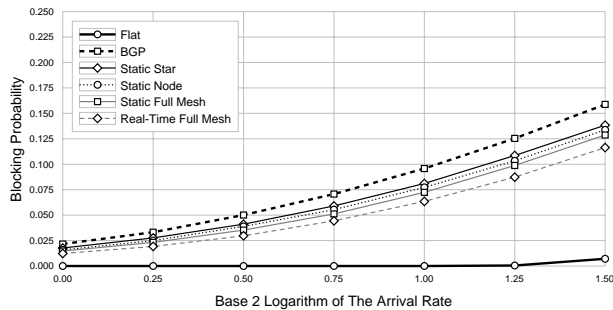


(h) End-to-end connection delay vs. arrival rate, 'B' topology

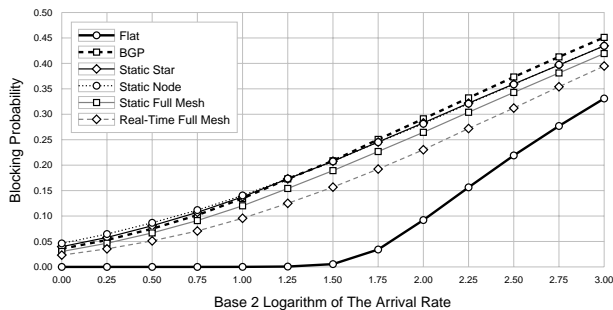
Fig. 2: Results for the persistent Per-Domain Backward Algorithm



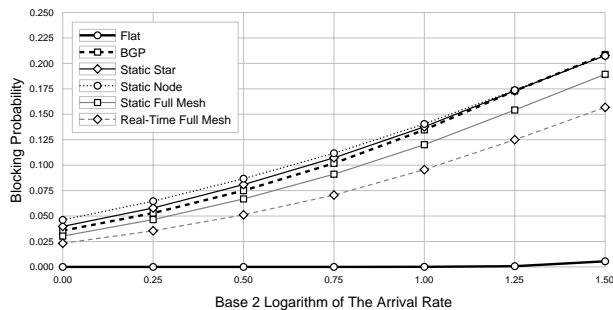
(a) Blocking probability vs. arrival rate, 'A' topology



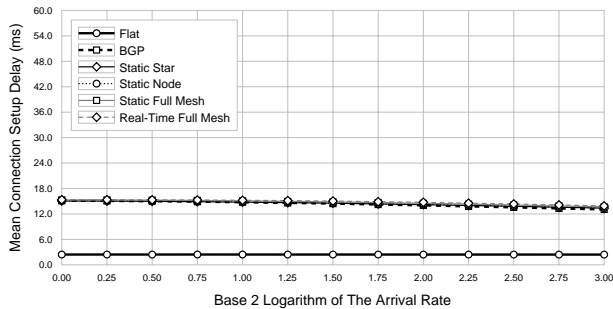
(b) Blocking probability vs. arrival rate, 'A' topology, magnified view



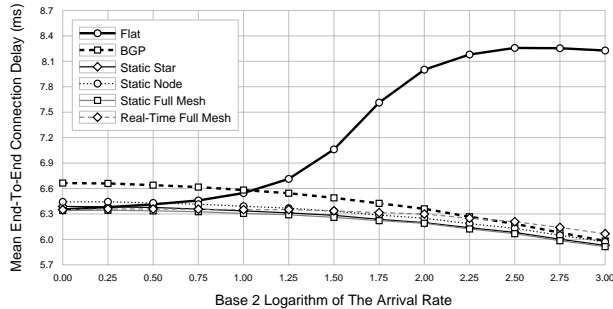
(c) Blocking probability vs. arrival rate, 'B' topology



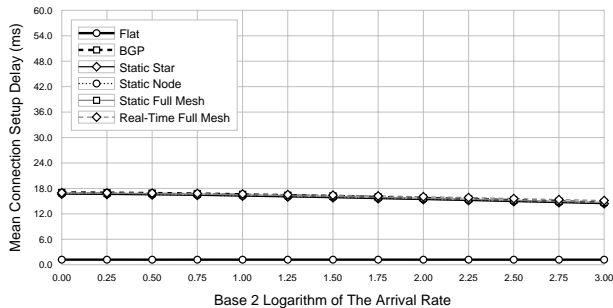
(d) Blocking probability vs. arrival rate, 'B' topology, magnified view



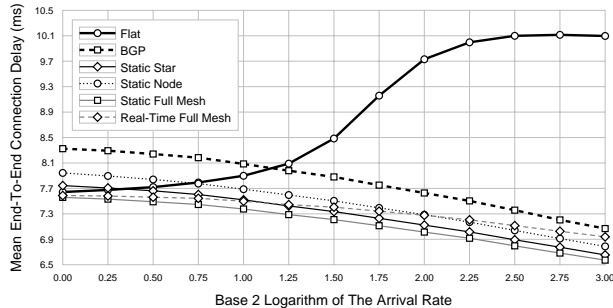
(e) Connection setup delay vs. arrival rate, 'A' topology



(f) End-to-end connection delay vs. arrival rate, 'A' topology

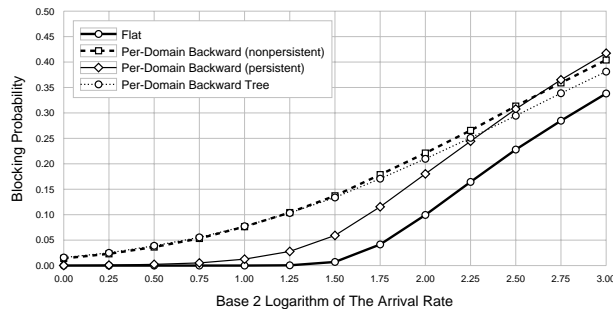


(g) Connection setup delay vs. arrival rate, 'B' topology

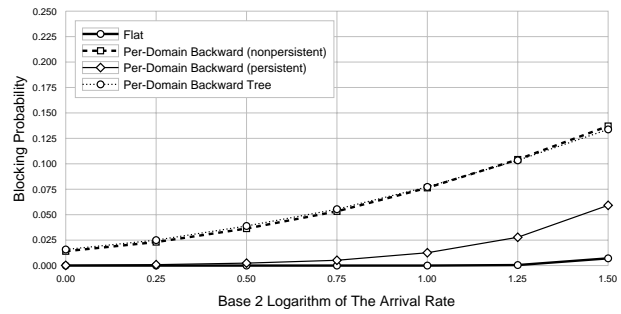


(h) End-to-end connection delay vs. arrival rate, 'B' topology

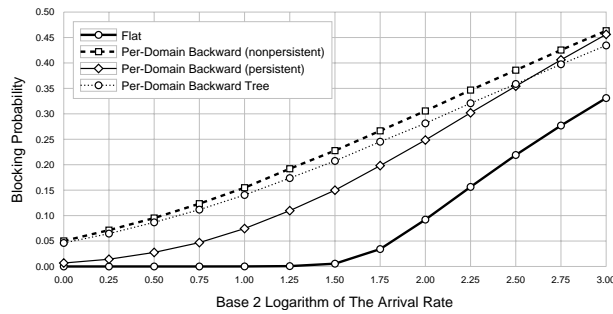
Fig. 3: Results for the Per-Domain Backward Tree Algorithm



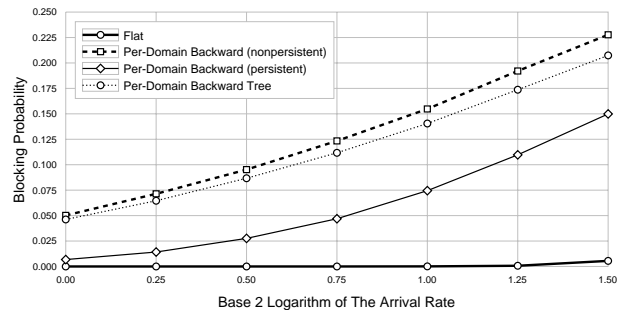
(a) Blocking probability vs. arrival rate, 'A' topology



(b) Blocking probability vs. arrival rate, 'A' topology, magnified view



(c) Blocking probability vs. arrival rate, 'B' topology



(d) Blocking probability vs. arrival rate, 'B' topology, magnified view

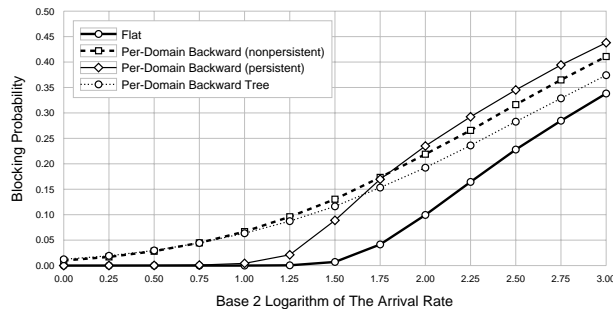
Fig. 4: Comparison of the blocking probabilities of the node-by-node path computation algorithms used in conjunction with the static nodal aggregation algorithm

paths returned by it will be longer, especially at higher loads (this is supported by the mean end-to-end connection delay statistics shown on Figures 2f and 2h). Now, since the computed AS paths are longer than the ones returned by the other AS path computation methods, the persistent Per-Domain Backward algorithm will fail more often.

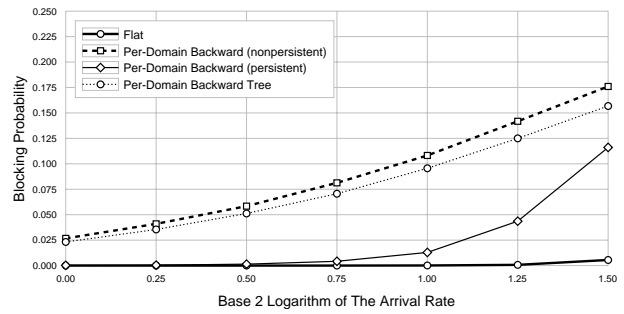
The static variation of the Full Mesh Aggregation-based algorithm shows slightly better performance than that of BGP. In general, it performs much worse than the real-time variation. However, as it keeps AS paths short, it does not cause problems when used in conjunction with the persistent Per-Domain Backward algorithm: in fact, the use of this method almost always results to node-by-node paths which are the shortest on the average (see Figures 1f, 1h, 2f, 2h, 3f, and 3h). Unfortunately, the relatively high computation cost of this procedure may not justify the small performance gains.

C. Star Aggregation

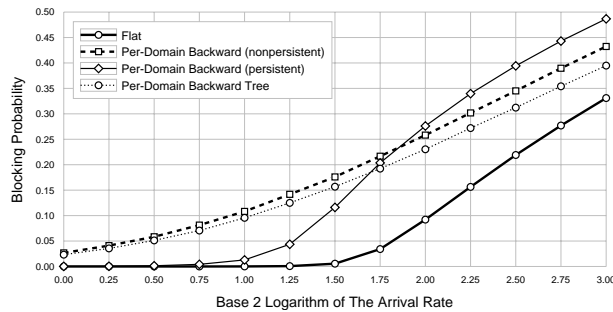
The performance of the Star Aggregation-based AS path computation algorithm is comparable to that of BGP. Depending on the topology, it may be slightly better (cf. Figures 1a, 1b, 2b, 2c, 3a, and 3b) or worse (Figures 1d and 3d). In general, it seems to perform better when the domains are relatively large (topology 'A'). Contrary to BGP, it is a QoS-aware algorithm. However, the minor performance advantage probably does not justify the increased computational complexity over BGP.



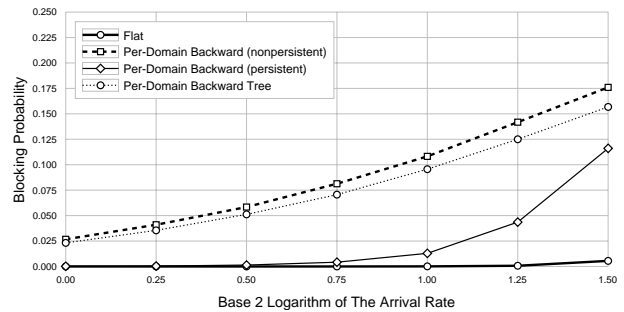
(a) Blocking probability vs. arrival rate, 'A' topology



(b) Blocking probability vs. arrival rate, 'A' topology, magnified view



(c) Blocking probability vs. arrival rate, 'B' topology



(d) Blocking probability vs. arrival rate, 'B' topology, magnified view

Fig. 5: Comparison of the blocking probabilities of the node-by-node path computation algorithms used in conjunction with the real-time full mesh aggregation algorithm

D. Nodal Aggregation

Curiously, although the Nodal Aggregation algorithm is much simpler than the Static Star Aggregation algorithm, it often seems to give comparable or sometimes even better results (cf. Figures 1a, 1b, 2a, 2b, 3a, and 3b). Performance is better when the domains are large (topology 'A'). Given the relatively low computational cost of this algorithm, it might provide a reasonable QoS-aware alternative to BGP.

E. Comparison of the node-by-node path computation algorithms

Figures 4 and 5 show that the blocking performance of the Per-Domain Backward Tree Algorithm is usually, but not always (see Figure 4b) better than that of the nonpersistent Per-Domain Backward Algorithm. More importantly, the difference between the two is usually not significant. Use of the persistent variant of the Per-Domain Backward Algorithm, however, consistently gives a lower blocking probability than the other two methods. Although we only show results for two AS-path computation algorithms, we have verified this to be the case for all of the AS-path computation algorithms studied in this paper. The price of this performance gain is an increase in the connection setup time (compare Figures 2e and 2g to Figures 1e, 1g, 3e, and 3g).

V. CONCLUSION AND FUTURE DIRECTIONS

Our results suggest that there is little to gain by using computationally complex, yet accurate routing algorithms both for AS-path and node-by-node path computation. Even the most complex and most accurate algorithms have failed to provide significantly better performance than BGP. The same also goes for the node-by-node path computation: the relatively simple Per-Domain Backward algorithm performs almost as well as the more convoluted Per-Domain Backward Tree algorithm, also known as Backward Recursive PCE-Based Computation (BRPC) algorithm in the literature.

We can also conclude that for the simpler AS path computation algorithms (i.e., BGP, static star, static node, and static full-mesh aggregation), the blocking probability of connections in the network is roughly proportional to the network utilization. This is not entirely true if the persistent variant of the Per-Domain Backward algorithm is used, which gives relatively low blocking under a certain connection arrival rate, then starts to block a large percentage of connections when the arrival rate reaches a certain threshold. It seems quite likely that no service provider will operate a network when the blocking probability is greater than 5%; for this reason, we may call the under 5% blocking range the “useful” range and conclude that the persistent variant of the Per-Domain Backward Algorithm consistently gives the lowest blocking probability.

The persistent variant of the Per-Domain Backward algorithm significantly improves over the blocking performance of the non-persistent algorithms, most importantly, the Per-Domain Backward Tree or BRPC algorithm, whose main weakness seems to lie in the fact that the computed path is not reserved until the whole computation process is completed, making the algorithm sensitive to interference from other connection requests. This improvement is quite pronounced in the low blocking probability range, irrespective of the AS-path computation algorithm or the network topology; also, the penalty paid in the form of increased connection setup delay is typically less than twice that of the original algorithms. Quite obviously, most applications are able to tolerate an increase from 15 msec to 30 msec in the connection setup time.

Another interesting observation is that trying to find an optimal AS path does not worth the effort. A relatively simple AS path computation algorithm (e.g. static nodal aggregation) combined with the persistent Per-Domain Backward algorithm outperforms the combination of the Real-Time Full Mesh Aggregation Algorithm and the Per-Domain Backward Tree Algorithm. In terms of computational complexity, the persistent Per-Domain Backward Algorithms is simpler than the Per-Domain Backward Tree Algorithm. For this reason, using BGP as the AS path computation algorithm seems quite a reasonable approach to inter-AS QoS provision. However, it can be argued that BGP is unaware of the QoS parameters of individual ASs and is therefore not a reasonable choice in cases where some ASs provide poor service. A comparably simple, yet more QoS-aware algorithm, like the static nodal aggregation might be more appropriate in such scenarios.

ACKNOWLEDGMENT

The authors would like to thank Dr. T. Ling for his helpful comments.

REFERENCES

- [1] *A Path Computation Element (PCE)-Based Architecture*, RFC 4655, 2006.
- [2] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh and J. van der Merwe, "The Case for Separating Routing from Routers," presented at the ACM SIGCOMM Workshop on Future Directions in Network Architecture, Portland, OR, Aug. 2004.
- [3] S. Shenker, "Fundamental Design Issues for the Future Internet," *IEEE J. Sel. Areas Commun.*, vol. 13, no. 7, pp. 1176–1188, Sept. 1995.
- [4] S. Amante et al. (2006, Nov.). Inter-Provider Quality of Service. MIT Commun. Futures Program. Cambridge, MA. [Online]. Available: <http://cfp.mit.edu/docs/interprovider-qos-nov2006.pdf>
- [5] *Internet Protocol Data Communication Service — IP Packet Transfer and Availability Performance Parameters*, ITU-T Standard Y.1540, 2007.
- [6] H. G. Perros. *An Introduction to ATM Networks*. New York, NY: Wiley, 2002.
- [7] H. G. Perros. *Connection-Oriented Networks: SONET/SDH, ATM, MPLS and Optical Networks*. New York, NY: Wiley, 2005.
- [8] T. Korkmaz and M. Krunz, "Multi-Constrained Optimal Path Selection," in Proc. IEEE INFOCOM 2001, vol. 2, pp. 834–843.
- [9] Z. Wang, "On the complexity of quality of service routing," *Information Processing Letters*, vol. 69, no. 3, pp. 111–114, Feb. 1999.
- [10] D. Yiltas and H. G. Perros, "A Composite QoS Metric for Multi-Attribute QoS-Based Multi-Domain Routing," Dept. Comput. Sci., NCSU, Raleigh, NC, 2009.
- [11] *A Backward-Recursive PCE-Based Computation (BRPC) Procedure to Compute Shortest Constrained Inter-Domain Traffic Engineering Label Switched Paths*, RFC 5441, 2009.
- [12] J.-P. Vasseur and J.-L. Le Roux, "Path Computation Element (PCE) Communication Protocol (PCEP)", Internet-Draft, draft-ietf-pce-pcep-09.txt, The IETF Trust, Nov. 2007.
- [13] G. Geleji, H. G. Perros, Y. Xin and T. Beyene, "A Performance Analysis of Inter-Domain QoS Routing Schemes Based on Path Computation Elements", presented at the 5th International Symposium on High Capacity Optical Networks and Enabling Technologies, Penang, Malaysia, Nov. 2008.