A Performance Analysis of Inter–Domain QoS Routing Schemes Based on Path Computation Elements

Geza Geleji and Harry G. Perros
Department of Computer Science
North Carolina State University
Raleigh, NC 27695–8206
ggeleji@ncsu.edu, hp@csc.ncsu.edu

Yufeng Xin
Renaissance Computing Institute
Chapel Hill, NC 27517
yxin@renci.org

Tsegereda Beyene
Cisco Systems, Inc.
tbeyene@cisco.com

***Abstract*—The Path Computation Element architecture has been proposed to separate forwarding and routing functionality in small–scale connection–oriented networks, such as MPLS, GMPLS, and ASON. If routing is performed by dedicated units, more sophisticated methods, such as multi–constraint QoS routing, may be implemented than currently widespread practices (i.e. OSPF, IS–IS, BGP, etc.). In this paper, we present a performance analysis of several distributed multi–domain QoS routing algorithms that may be implemented on the PCE platform.**

I. Introduction

It has been argued that the Internet should perform admission control in order to support applications with hard real–time QoS requirements, see, for instance, Shenker [15]. Feamster et al. [8] have proposed the Routing Control Platform to separate inter–domain routing from forwarding. They claim RCP is incrementally deployable and provides benefits even if not deployed across the whole network, such as the facilitation of traffic engineering, simpler policy expression, and enforceable consistency of routes.

The Path Computation Element (PCE) architecture (RFC 4655 [7]) has been recently proposed as a new approach to deal with the challenges posed by constraint–based path computation in single–domain or small multi–domain MPLS, GMPLS, ASON or similar networks for the purposes of Quality of Service (QoS) provision. The architecture completely separates routing and forwarding by designating Path Computation Elements (PCEs) in the network, in accordance with the refactoring of the IP control plane proposed by Rexford et al. [13]. In this paper, the authors propose to factor IP control functionality into two planes, the dissemination plane, whose "primary objective is the timely and reliable dissemination of information to and from network elements", and the decision plane, which should "make all decisions driving network behavior, including reachability, routing, access control, security, and interface configuration". PCE units constitute an inter–domain path computation plane [14], whose sole responsibility is to supply routers and hosts (i.e. the forwarding plane) with paths fulfilling a desired set of QoS constraints. The advantage of this approach is that PCE nodes will have an up–to–date, domain–wide view of the network state while still relying on distributed algorithms, and therefore, they will be able to make routing decisions subject to QoS constraints more effectively. Also, both the routing and forwarding tasks may be performed by separate, more specialized equipment. Proponents of the architecture suggest that it will facilitate the implementation of more complex path computation schemes. Rexford et al. [13] also argued that the approach will simplify the IP control plane. The PCE architecture supports mechanisms that maintain the confidentiality of routing information pertaining to a domain (see Bradford et al. [6]). In general, according to Boucadair et al. [5], such functional decomposition is desirable when trying to achieve a business–process view of various tasks to be performed by network operators. Disadvantages include the possibly high delays in connection establishment if each connection request triggers a path computation procedure (Feamster et al. [8]), and the architecture's supposed inability to deal with the requirements of large–scale networks, like the Internet (Farrel et al. [7]).

Unfortunately, there is relatively little research reported in the open literature on the implementation and performance of PCE–based routing and as a consequence, many aspects of the standard have not been worked out yet. The most important of these is how the PCEs cooperate with each other: PCEs may form a large, distributed, possibly redundant system dedicated to serving path computation requests as efficiently as possible. Such a system needs to have an up–to–date, easily accessible database of available network resources (referred to as the Traffic Engineering Database in the PCE documents). Since this is not possible in the general case, certain restrictions have to be made. For instance, the authors of the original RFC, Farrel et al. [7], suggest limiting the range of action of the path computation system to a few domains, and the PCEs may only be used if both endpoints of a requested path lie within these domains. This limits the algorithmic complexity and decreases connection set–up times. Efficient multi–constrained routing still constitutes a significant challenge (McAuley et al.

Corresponding author: Geza Geleji

[12]), especially across domains (Griffin et al. [10]). A good example of a bridging solution is the meta–QoS–class plane–based approach (Levis et al. [11] and Griffin et al. [10]), which relies on class descriptions such as "very low one–way transit delay" and "low delay and any packet loss rate" and leaves the implementation details to the provider. Unfortunately, this scheme does not support hard end–to–end QoS guarantees. The PCE–based architecture, in general, is not expected to provide optimal paths (Yannuzzi et al. [18]), instead, it tries to reduce the challenges posed by the multi–domain (Yannuzzi et al. [18] and Aslam et al. [3]) and multi–constraint (McAuley et al. [12]) requirements, especially when hard QoS guarantees are desired, such as end–to–end delay, jitter and packet loss rate.

In the paper, we use simulation to investigate the feasibility of PCE–based QoS routing schemes in small multi–domain networks by measuring blocking probability, network utilization, routing efficiency in terms of path cost, and connection set–up times. We study the performance of simple, sub–optimal, distributed path computation algorithms, comparing the results to ideal solutions, such as Dijkstra's algorithm executed on the whole network. The algorithms presented herein are based on ideas presented by Aslam et al. [3] (e.g. the Per–Domain methods rely on the loose route refinement detailed in their paper) and Vasseur et al. [17] (the Per–Domain Backward Tree method closely resembles BRPC).

The paper is organized as follows. In the next section, we first describe the main features of the simulator, then the path computation algorithms. The results are presented in Section III-B, and the conclusions are summarized in Section IV.

II. The PCESIM simulator

We have developed a discrete event simulator, referred to as PCESIM, which simulates the exchange of control messages, such as connection set–up requests, path computation requests and responses between Path Computation Clients (PCC) and PCEs, inter–PCE messages (Path Computation Request, Path Computation Reply) and error notifications, as discussed in the PCEP documents [2], [16]. We simulated a multi–domain network consisting of physical nodes interconnected by unidirectional links. Each node belongs to exactly one domain and is assigned a pair of geographic coordinates, based on which, the message propagation delay along a link is calculated, assuming that links lie on a great circle on the surface of a sphere approximately the size of the Earth. Links have a pre–defined capacity from which bandwidth is allocated exclusively to individual connections based on their QoS requirements. We assume that each node has enough capacity to process all the packets, even if all links are fully utilized. We also assume that each domain contains at least one PCE attached to one of its nodes.

The simulator takes a description of the network topology and a sequence of connection set–up requests to be executed on the network as input. The requests are processed on a First–Come–First–Served basis. The simulator was written in C++ and an automatically generated documentation of the PCESIM source code is available on–line [9].

A. The PCE–based Inter–Domain QoS Routing Schemes

A connection set–up request provides the following information: source node, destination node, bandwidth requirements, desired QoS class, and desired QoS parameters. In this paper, we only consider the top priority QoS class defined for sources which transmit continuously at a constant bit rate and the information transmitted is time sensitive. Examples are unencoded video and circuit emulation. For this type of service, the only bandwidth that needs to be specified is the maximum transmission rate. This is the bandwidth that needs to be allocated on each link along the path. This QoS class typically receives top priority in the output pot buffer of a router. Therefore, in theory, packets belonging to this class will never suffer queueing delay before they are transmitted out. This means that the end–to–end delay is equal to the propagation delay plus the time spent in each router along the path. The latter component is very small compared to the propagation delay, and therefore, is ignored in this study. As for the packet loss, we assume that it is negligible because this QoS class will more likely have a high space priority.

In view of the above assumptions, the inter–domain algorithms studied in this paper simply attempt to minimize the cost which is the propagation delay.

All of the three procedures rely on a pre–computed AS path, i.e. a sequence of domains through which the path will be routed (this sequence is supplied to the simulator as input). We assume that control messages are sent through control channels over the same IP data plane used to transmit the packets of each connection. These control channels are assumed to have sufficient capacity for transmitting control messages, and they do not interfere with the bandwidth used for the IP data plane. Each PCE communicates with any other PCE in the network over the shortest possible path.

We have implemented the following path computation algorithms:

1) Per–Domain Backward Method: A PCE executing this algorithm forwards each incoming path computation request to a PCE in the last domain of the path, which we will refer to as the "end PCE". Since PCEs are aware of the interdomain links to all adjacent domains, the end PCE finds the cheapest path from the destination node to the domain before the last and allocates the requested amount of bandwidth along the newly computed path segment, which is terminated at an egress node, call it A, of the domain before the last. When the path segment has been set up, the last PCE forwards the request to a PCE in the domain before the last, whose job is to find the cheapest path from an egress node of the preceding domain, 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 destination node. This procedure is repeated until the first (source) domain is reached, where a PCE finds a path segment from the requested 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 shortest paths within domains given that the egress node has been fixed. In view of this, it is unlikely that it will find the shortest path between the source and the destination; however, its relatively low computational complexity and communicational overhead make it a promising candidate.

2) The Per–Domain Ping–Pong Method: This method relies on the same principle as the previous algorithm. However, a PCE receiving a setup request, instead of forwarding it to the last domain, will first find the shortest path to an ingress node of the second domain, which will, in turn, find the shortest path to an ingress node of the third domain, and so on, until the last domain is reached. As opposed to the above algorithm, instead of reserving capacities along the path segments immediately after the segment has been computed, reservations are done separately after the path computation process has been completed, starting from the end PCE and moving backwards towards the originating PCE. The name "ping–pong" intends to illustrate the fact that path computation happens in the forward direction, and reservation goes backwards along the same path). Also, the per–domain backward method uses the shortest path to forward the setup request from the first PCE to the last one. In the per–domain backward path computation algorithm, the request is sent from the first PCE to the last one along the shortest path and then travels backwards from PCE to PCE. In the ping–pong method, the request travels from PCE to PCE in both directions, thus taking a longer time to set up the connection. As the reservations are not done immediately after the path computation, individual requests have to contend for resources. On failure of allocating a resource, all associated resources are freed immediately.

3) The Per–Domain Backward Tree Method: This algorithm tries to optimize the cost of the computed path by finding trees of paths from a node to a given neighbor domain. It is similar to the per–domain backward method, except instead of propagating a sub–optimal path from the destination node towards the source, a tree of all available paths is forwarded. This is an implementation of the Backward Recursive Path Computation (BRPC) algorithm presented by Vasseur et al. [17], with one minor difference, namely, the setup request is forwarded along an optimal path from the PCE in the source domain to the PCE in the destination domain. The original BRPC method forwards the request from one PCE to a PCE in the next domain, which implies a sub–optimal path and additional processing delays. As the tree travels along the AS path in reverse direction, a PCE in each domain extends it by adding a set of paths from the egress nodes of the preceding domain 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 backwards method. 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 method does not necessarily find the optimal path either because it is constrained by the pre–determined AS path which may not be optimal in itself. Also, the algorithm will not be able to find an existing path if it does not comply with the specified AS path (e.g. there is a path available along the domain sequence 1–2–3–4–3–4–5, but the specified constraint is 1–2–3–4–5). Note that in our implementation, the PCE responsible for path selection will select the minimum cost path from the tree. If that path has become unavailable due to other connections, the path computation process will fail and no other paths from the tree are considered.

4) Reference Algorithms: To provide a basis of comparison for the performance of these algorithms, we have also implemented three reference algorithms. In all cases, we assume that path computation happens instantaneously and no control messages are sent between PCEs. The first one, referred to as the *Flat Path Computation* method, is Dijkstra's shortest path algorithm applied to the whole network. Results from this algorithm will serve as theoretical bounds for all other algorithms.

Our second reference algorithm, called *Instantaneous Per–Domain Backward Method*, is the per–domain backward method, except that execution does not involve control message exchanges and, therefore, takes place instantaneously. Results from this method are expected to provide a bound on the per–domain backward and ping–pong algorithms.

The third reference procedure, referred to as the *Instantaneous Per–Domain Backward Tree Method*, is similar to the per–domain tree–based method without message exchanges, that is, execution of the algorithm takes place instantaneously.

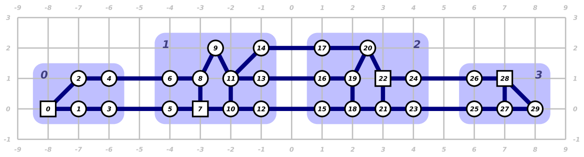
III. Simulation Environment and Results

We have conducted several experiments using the algorithms discussed above on various test networks presented below. In this section we present some of our most illustrative results along with the details of the simulation environment.

A. Test Networks

Our experiments have been conducted on five different test configurations, each of which uses a test network derived from the ones presented in Figure 1. Both maps are assumed to be on the surface of a sphere with a radius of 6372.8 km, thus, grid divisions correspond approximately to geographic latitudes and longitudes with a distance of approximately 111 km. In all experiments, the capacity of each link is 10 units and the signal propagation speed is assumed to be 200 km/ms. The shaded areas constitute domains. Nodes represented by a square contain one PCE and nodes represented by circles do not. Each domain has exactly one PCE.

Figure 1a shows a *linear* test network, which consists of a sequence of domains. In this network, the AS path can be easily and uniquely determined for any connection request. Through this fact, the number of possible paths that may be assigned to any particular connection is very limited (they all traverse the same AS path), and, as a consequence, we expect the optimal and suboptimal path computation methods


(a) Linear

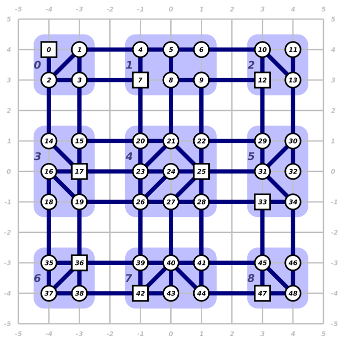

(b) Square Mesh

Fig. 1: Test Networks

to behave similarly. We will also use two variations of this network: the *vertically stretched* and the *horizontally stretched* linear networks. The former was obtained by moving nodes 2, 4, 6, 8, 9, 11, 13, 14, 16, 17, 19, 20, 22, 24, 26 and 28 (the top two rows in Figure 1.a) North by 20°, a distance of approximately 2000 km. The latter is a result of increasing the length of all interdomain links from 2° to 20° (again, the displacement is roughly 2000 km).

The *square mesh* test network (see Figure 1b) consists of domains with a higher degree of connectivity to other domains. We expect this network to demonstrate the differences in routing efficiency between near–optimal (e.g. the flat method and the tree–based methods) and sub–optimal methods (i.e. per–domain) to a greater extent than the linear network. We have also studied a slight variation of this network: the *full mesh domains* version, in which, as the name implies, each domain is augmented to a full mesh while the inter–domain connections remain unaltered.

B. Numeric Results

The connection setup requests were generated by a Poisson process with inter–arrival times ranging from $1/16$ to 256 milliseconds. The inter–arrival time is the horizontal axis of all of the graphs presented in this paper. The holding time, i.e. the time a connection is up, follows an exponential distribution with a mean of 4 ms, and the requested bandwidth is a uniformly distributed integer with a minimum of 1 and a maximum of 10 units. Each experiment consists of 250000 connection requests.

For each experiment, we calculate the following performance metrics: the blocking probability (percent of blocked connection requests), the mean network utilization (the time average of link utilization averages for all links), the mean cost (i.e. propagation delay) of the paths assigned to each successfully routed connection, the mean length or hop count of the successfully routed connections, the mean admission delay, i.e. the mean time required to successfully set up a connection (including the time for resource allocation), and the mean rejection time, i.e. the mean time required to determine that the connection can not be admitted. For high values of the inter–arrival time, most of the connections will be admitted, and, as a consequence, the sample sizes for measuring the rejection delay will be small, causing high variation. However, most data points represent the mean of the sampled random variable with a very narrow confidence interval, which is not shown on the graphs, since it is not discernible.

Note that the range of inter–arrival times used includes values significantly smaller than the mean holding time. Though these cases do not represent real–life situations, we have included them to demonstrate the behavior of the algorithms under very high traffic loads.

The colors used in the diagrams correspond to the routing algorithms as follows: Flat Path Computation (gray), Instantaneous Per–Domain Backward Method (blue), Per–Domain Backward Method (red), Per–Domain Ping–Pong Method (black), Per–Domain Backward Tree Method (orange), Instantaneous Per–Domain Backward Tree Method (green).

1) Test Configuration No. 1: the unmodified linear network: Looking at the chart of the blocking probability (Figure 2a), one can see a pronounced difference between the real and the reference routing algorithms, which is due to the fact that the reference algorithms do not exchange control messages and therefore, they execute instantaneously. The delay resulting from message exchanges has a significant effect on blocking; the longer a connection setup takes, the more likely it is to be blocked as the amount of available resources may change during the setup process. However, the difference between the optimal and sub–optimal reference algorithms (gray curve versus blue and green) is minimal, illustrating the fact that certain networks are capable of achieving a high routing efficiency without an optimal routing algorithm. Among the distributed algorithms, the per–domain tree–based method (orange) gives the best performance. The per–domain ping–pong method is the worst, because it relies on sub–optimal control paths in both directions (note that the other two forward the connection setup request to the destination node along an optimal control path). The per–domain backward method (red) lies between the other two; its performance is not very good at high loads, but improves as the load decreases. For low levels of traffic, it outperforms the per–domain tree–based method. This could be attributed to the fact that it starts to make the resource reservation as soon as the intra–domain path segments are identified, while the per–domain tree–based method only reserves the capacity when the complete path tree has been calculated. However, the difference due to this effect is negligible.

Note that the per–domain backward method achieves the same level of blocking as the per–domain ping–pong method at a higher level of resource utilization (see Figure 2b). This is due to the fact that, on the average, it routes connections along longer paths (see Figure 2d; the graph depicting the mean path length in number of hops is, qualitatively speaking, almost indistinguishable). It appears that the per–domain ping–pong method favors connections that may be routed over shorter paths. This fact may be verified by looking at Figure 2c, which shows the distance fairness of these two algorithms and of the flat method (gray line), which has been included for comparison purposes, when the mean inter–arrival time is 1/4 (thick, solid lines and circles) and 2 (thin, dotted lines and diamonds). Each data point represents the ratio of blocked calls for which the shortest possible path (in an empty network) between the endpoints has a cost (i.e. delay) in the range $[(x-10k), x]$, where x is given on the horizontal axis in milliseconds. The per–domain backward method (red) admits a greater number of long–distance connections, conversely, the per–domain ping–pong method favors short–distance connections. As the load is decreased (dotted lines), the flat method shows a much higher degree of fairness, while the same can not be said for the Per–Domain Ping–Pong method. The Per–Domain Backward method lies in between the two and it still favors short–distance connections.

Figure 2d shows the mean path costs of each of the algorithms. As the traffic load decreases, the algorithms converge to a path cost which reflects their routing efficiency. Based on this value, two basic classes may be distinguished: the class of *near–optimal* and the class of *sub–optimal* methods. The flat path computation (gray) and the tree–based methods (green and orange) constitute the former, the rest the latter. Obviously, for this topology, the tree–based methods perform nearly as well as the flat method, while all of the per–domain methods show some deficit. This is true even for very low loads, which suggests that the reason may be the sub–optimality of the path computation algorithms. It is interesting to note that for high loads, the optimal flat path computation method (gray) and the instantaneous per–domain method (blue) do not differ significantly neither in terms of blocking, nor in terms of routing efficiency. As will be seen below, where other topologies are studied, this due to the network topology rather than the path computation methods.

Finally, Figure 2e shows the mean rejection delay (in msecs) for the distributed algorithms that rely on control message exchanges. This is the mean time it takes until a connection is rejected due to lack of available resources. The per–domain ping–pong method has the lowest values starting at about 4.5 when the mean inter–arrival time is $1/16$ and monotonically increases to 8.8 when the mean inter–arrival time is 256. The per–domain tree–based method shows values that are a little higher. Since the per–domain backward method (red) and the per–domain tree–based method (orange) admit connections using similar mechanisms, their admission delays do not differ significantly. The minimal difference present may be explained by differences in routing efficiency and blocking characteristics. The somewhat lower delay of the per–domain ping–pong method can not be attributed to better performance, but rather to the fact that this algorithm routes a higher ratio of low–distance connections (see the discussion for Figure 2c above). Note that the per–domain ping–pong method relies on sub–optimal control paths in both directions, while the other two use an optimal path in the forward direction.

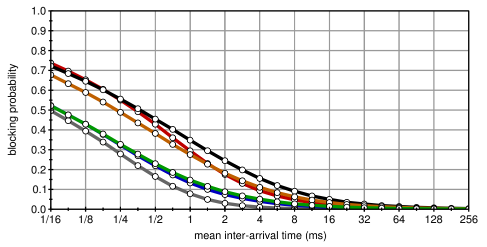
On average, the per–domain tree–based method (orange) takes a lot longer to reject connections than the per–domain backward method does. This is a direct consequence of the tree–based method having to compute first a full tree of end–to–end paths before it can reserve resources (and determine failure or success). Since the per–domain backward method reserves each intra–domain segment as soon as it has been computed, it will immediately fail during computation if a segment can not be set up through a particular domain. The per–domain ping–pong method shows a significantly lower delay, which, again, is not likely to be attributed to better performance, but rather to the distance fairness characteristics.

2) Test Configuration No. 2: the vertically stretched linear network: Vertically stretching the linear network increases the performance gap between the three distributed algorithms and their reference counterparts. For instance, the gap between the two groups is increased from the value shown in Figure 2a (approximately 0.2) to about 0.3. Some aspects of performance (e.g. mean path lengths in hop–counts) do not change significantly because of this topology modification, while some (e.g. mean admission delays) retain the same characteristics. These graphs are not presented in this paper. The mean rejection delay graph (see Figure 2f) supports the hypothesis that the mean rejection delay of the per–domain ping–pong method (black) is generally not lower than that of the per–domain backward method (red). In this case, for mean inter–arrival times above 2, the former has a significantly higher mean rejection delay.

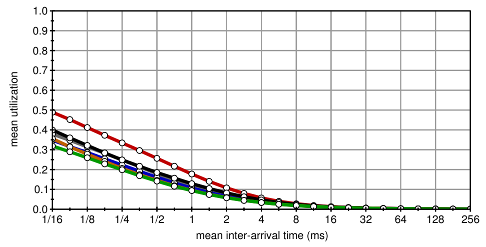
3) Test Configuration No. 3: the unmodified square mesh: Most of the measurements on the square mesh network yielded significantly different results from those obtained on the linear networks. The blocking characteristics (Figure 2g) show that the flat path computation method (gray) is more efficient than any of the per–domain methods (this difference was not observable on the linear networks). The distributed methods (red, orange and black) are still quite closely grouped together on the graph. We are omitting the graphs of the mean path costs and lengths, as well as the mean admission and rejection delays.

4) Test Configuration No. 4: square mesh with full mesh domains: Increasing the connectivity of the routers in each domain seems to impact the performance of the per–domain backward method and the per–domain ping–pong method relative to each other (compare Figure 2g to 2h). This time, the distance fairness problems of the per–domain ping–pong method seem to have diminished; in fact, it even shows better blocking performance than the per–domain backward method. The mean path costs and lengths of the two are very close to each other as well as the mean admission delays (omitted). The mean rejection delay of the per–domain backward method is somewhat below that of the per–domain ping–pong algorithm.

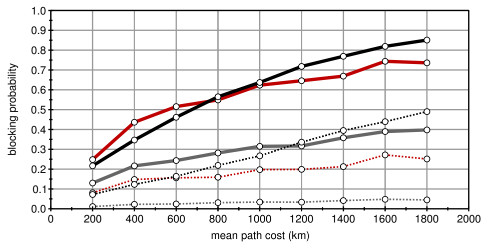
5) Test Configuration No. 5: the horizontally stretched linear network: Stretching the linear test network horizontally emphasizes the difference between the per–domain tree–based



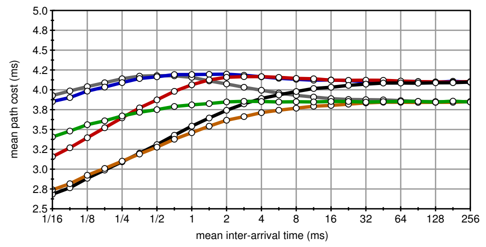
(a) Blocking probability, config. 1



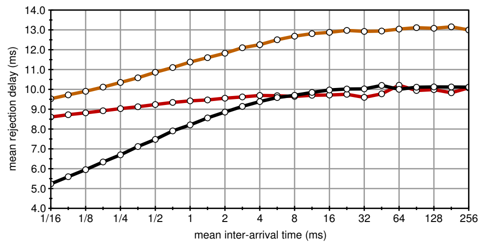
(b) Utilization, config. 1



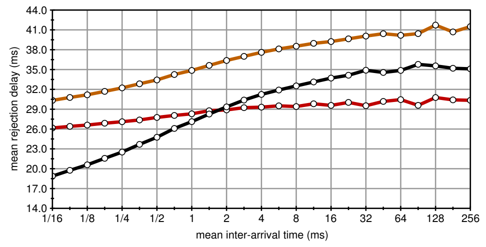
(c) Distance fairness, config. 1



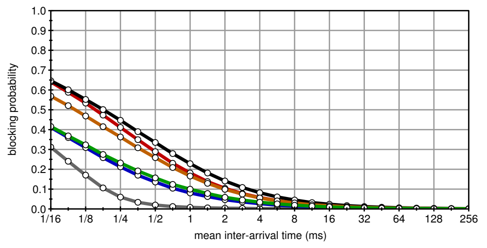
(d) Mean path cost, config. 1



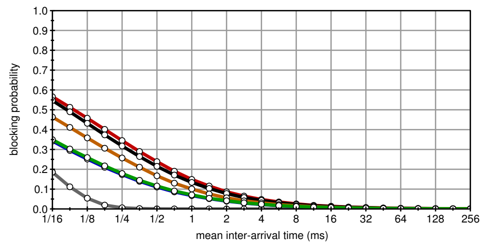
(e) Mean rejection delay, config. 1



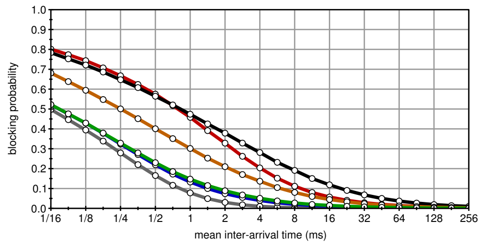
(f) Mean rejection delay, config. 2



(g) Blocking probability, config. 3



(h) Blocking probability, config. 4



(i) Blocking probability, config. 5

Fig. 2: Simulation results

method (orange) and the other two distributed algorithms (red and black; see Figure 2i). The distance fairness characteristics of the per–domain backward and ping–pong methods have not changed. Though the performance differences between the distributed and the simulated algorithms are still quite high, the per–domain tree–based method (orange) seems to narrow this gap.

IV. Conclusion

In this paper, we have presented a performance analysis of three distributed path computation algorithms designed for the PCE architecture on various test networks. Our results indicate that the per–domain tree method (which is similar to the BRPC algorithm [17]) is a promising candidate. Its only shortcoming appears to be the somewhat higher connection rejection delay. We have also found that the performance differences between the algorithms greatly depend on the network topology; e.g. for a linear network, per–domain methods are nearly as good as optimal algorithms. This fact may suggest that instead of trying to use optimal algorithms for routing, we should optimize the topology using off–line optimization.

References

[1] S. Amante et al., *Inter–provider Quality of Service*, white paper draft, version 1.1, Quality of Service Working Group, MIT Communications Futures Program, November 2006

[2] J. Ash, J.–L. Le Roux, *Path Computation Element (PCE) Communication Protocol Generic Requirements*, RFC 4657, The Internet Society, September 2006

[3] F. Aslam, Z. A. Uzmi, A. Farrel, *Interdomain Path Computation: Challenges and Solutions for Label Switched Networks*, IEEE Communications Magazine, vol. 45, no. 10, pp. 94–101, October 2007

[4] T. Beyene, Y. Xin, M. Turabi, K. Raza, *PCE Based Grid Networking*, IEEE Symposium on Computers and Communications (ISCC), Aveiro, Portugal, 1–4 July 2007

[5] M. Boucadair, P. Lévis, D. Griffin, N. Wang, M. Howarth, G. Pavlou, E. Mykonati, P. Georgatsos, B. Quoitin, J. Rodríguez Sánchez, M. L. García–Osma, *A Framework for End–to–End Service Differentiation: Network Planes and Parallel Internets*, IEEE Communications Magazine, vol. 45, no. 9, pp. 134–143, September 2007

[6] R. Bradford, J.–P. Vasseur, A. Farrel, *Preserving Topology Confidentiality in Inter-Domain Path Computation Using a Key-Based Mechanism*, Internet–Draft, draft-ietf-pce-path-key-03.txt, The IETF Trust, May 2008

[7] A. Farrel, J.–P. Vasseur, J. Ash, *A Path Computation Element (PCE)–Based Architecture*, RFC 4655, The Internet Society, August 2006

[8] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, J. van der Merwe, *The Case for Separating Routing from Routers*, ACM SIGCOMM Workshop on Future Directions in Network Architecture, Portland, OR, August 2004

[9] G. Geleji, *PCE–Based Multi–Domain QoS Routing Simulator (developer's documentation)*, on–line resource available from `http://andromeda.csc. ncsu.edu/pcesim/`, accessed 18 March 2008

[10] D. Griffin, J. Spencer, J. Griem, M. Boucadair, P. Morand, M. Howarth, N. Wang, G. Pavlou, A. Asgari, P. Georgatsos, *Interdomain Routing through QoS–Class Planes*, IEEE Communications Magazine, vol. 45, no. 2, pp. 88–95, February 2007

[11] P. Levis, M. Boucadair, P. Morand, J. Spencer, D. Griffin, G. Pavlou, P. Trimintzios, *A New Perspective for a Global QoS–based Internet*, Journal on Communications Software and Systems, vol. 1, no. 1, pp. 13–23, September 2005

[12] A. J. McAuley, K. Manousakis, L. Kant, *Flexible QoS Route Selection with Diverse Objectives and Constraints*, 16th International Workshop on Quality of Service (IWQoS), Enschede, The Netherlands, 2–4 June 2008

[13] J. Rexford, A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, G. Xie, J. Zhan, H. Zhang, *Network–Wide Decision Making: Toward a Wafer–Thin Control Plane*, HotNets–III, San Diego, CA, November 2004

[14] F. Ricciato, U. Monaco, D. Ali, *Distributed Schemes for Diverse Path Compu-tation in Multidomain MPLS Networks*, IEEE Communications Magazine, vol. 43, no. 6, pp. 138–146, June 2005

[15] S. Shenker, *Fundamental Design Issues for the Future Internet*, IEEE Journal on Selected Areas in Communications, vol. 13, no. 7, pp. 1176–1188, September 1995

[16] J.–P. Vasseur, J.–L. Le Roux, *Path Computation Element (PCE) Communication Protocol (PCEP)*, Internet–Draft, draft–ietf–pce–pcep–09.txt, The IETF Trust, November 2007

[17] J.–P. Vasseur, R. Zhang, N. Bitar, J.–L. Le Roux, *A Backward Recursive PCE-based Computation (BRPC) Procedure To Compute Shortest Constrained Inter-domain Traffic Engineering Label Switched Paths*, Internet–Draft, draft-ietf-pce-brpc-09.txt, The IETF Trust, April 2008

[18] M. Yannuzzi, X. Masip–Bruin, S. Sánchez, J. Domingo–Pascual, A. Orda, A. Sprintson, *On the Challenges of Establishing Disjoint QoS IP/MPLS Paths Across Multiple Domains*, IEEE Communications Magazine, vol. 44, no. 12, pp. 60–66, December 2006