Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT

Computer Communications 26 (2003) 1764–1774

computer communications

www.elsevier.com/locate/comcom

Practical QoS network system with fault tolerance

S.S. Lee*, S. Das, H. Yu, K. Yamada, G. Pau, M. Gerla

Computer Science Department, University of California, Los Angeles, CA 90095-1596, USA

Abstract

In this paper, we present an 'emulation environment' for the design and planning of QoS-enabled IP networks. Specifically, QoS support for real-time traffic in future IP networks require 'fault tolerance' for mission-critical applications such as tele-medicine. Based on this premise, our goal is to design the QoS network system for the traffic management so that QoS requirements are satisfied. Routing is a key component of this architecture, in which it enables the efficient use of the provisioned bandwidth. Our approach is to make routers in the network capable of finding QoS paths in a distributed manner in response to changing traffic demands. As an extension of QoS routing, each router is able to compute 'multiple QoS paths' to the same destination. The multiple QoS paths are utilized in parallel making the entire network system less prone to network failures. In computing multiple QoS paths, practical issues such as computation latency and resource requirement must be considered. With the use of multiple QoS paths, our emulation environment permits us to design more reliable QoS network systems and to test various applications for feasibility before deployment. This paper presents the multiple QoS algorithms with notable enhancements and the system architecture for fault tolerance. The relevant experiment results have also been presented.

Keywords: QoS routing; Multiple paths; Fault tolerance; Emulation environment

1. Introduction

The importance of QoS guarantees in IP networks for the currently emerging and future network applications has been discussed for many years, and substantial research contributions have been made accordingly. Following such a significant research stream, in this paper, we present an enhanced QoS-compliant network testbed that can serve as a QoS-enabled IP networks design and planning tool.

In order to provide a seamless multimedia traffic support, the underlying network system must guarantee QoS. Network systems for providing QoS guarantees may consist of various building blocks such as resource assurance, service differentiation, and QoS routing. Among these aspects, the main research issue addressed in this paper is QoS routing; namely, the capability of finding network paths satisfying given multiple QoS constraints. This is fundamental groundwork for the other QoS components. Pioneering research about the aspect was started by Guerin et al. and Cavendish and Gerla [1,2]. The latter showed a possible use of the Bellman–Ford algorithm in the link state (OSPF) routing environment for QoS support. These early contributions clearly pointed that a set of multiple QoS constraints (e.g. bandwidth, delay, reliability, etc.) must be simultaneously satisfied for QoS-sensitive applications, forming a body of meaningful specifications for practical approaches [3]. Along with these foundations Dubrovsky et al. [4], showed the practicality of such QoS routing algorithms by applying them to a network environment for IP Telephony. QoS routing issues were further accelerated by the development of MPLS [5] which deploys fast packet-forwarding mechanisms in IP core networks.

In this paper, we review the OSPF-based QoS architecture with the above models and we show how it can be applied to satisfy given QoS constraints. Moreover, we will emphasize an additional issue that is often disregarded in conventional QoS routing architectures: robust QoS guarantee in an unreliable, unpredictable network environment [6,7]. Network reliability is critical for mission-critical applications (e.g. tele-medicine and distance surgical operations), the applications require not only reasonable QoS guarantees but also transparent reliability/robustness from failures or changes in the underlying network. For this purpose, an effective approach for reliable QoS support was recently introduced in Ref. [8], which effectively computes multiple QoS paths with minimally overlapped links.

* Corresponding author. Tel.: +1-310-206-6518; fax: +1-310-825-7578.
E-mail addresses: sslee@cs.ucla.edu (S.S. Lee), shanky@cs.ucla.edu (S. Das), heeyeoly@cs.ucla.edu (H. Yu), kenshin@cs.ucla.edu (K. Yamada), gpau@cs.ucla.edu (G. Pau), gerla@cs.ucla.edu (M. Gerla).

0140-3664/03/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0140-3664(03)00049-5

By provisioning multiple QoS paths, the network system can provide backup paths when one or more paths are detected as corrupted. Besides, the spreading of network traffic over the provisioned multiple QoS paths favors even network resource utilization. Benefits of provisioning multiple QoS paths are illustrated in Refs. [8,9].

With such a fault-tolerant QoS-compliant architecture as a target, we have deployed a QoS testbed that will assist in the implementation, evaluation, and comparison of various architecture components. In this paper, we present the practical QoS system equipped with an effective QoS routing algorithm for multipath and fault tolerance implementation. In Section 2, the core QoS routing algorithms are thoroughly introduced and variants are discussed. Section 3 depicts the entire network system architecture. Section 4 presents experiment results obtained with the proposed network system and demonstrates its effectiveness in representative traffic scenarios.

2. Multiple QoS routing algorithms

As emphasized in Ref. [7], the reliability aspect must be highlighted in the QoS provisioning. In order to achieve the reliable QoS services, we propose provisioning multiple QoS paths as originally introduced in Ref. [8]. The approach takes advantage of multiple alternate paths in the face of network failures and utilizes network resources more evenly. The scheme showed significant improvements of fault tolerance. [8] also showed various path management schemes with which multimedia traffic packets or flows can be spread over multiple paths for different load balancing effects. In addition, provisioning multiple paths also shows various positive effects. For instance, most QoS routing approaches are required to have global link state information as the entire map of the network. This global link state information is obtained via the OSPF flooding mechanism [10], and it needs to be refreshed frequently. Such information acquisition through frequent flooding is an expensive process. Lee and Pau [9] showed the efficacy of multiple paths on the problem of this scalability issue. Besides, using multiple paths in parallel utilizes the network resources more evenly (i.e. load balancing), diffuses the dynamic network traffic changes, and soothes frequent network flooding. Regarding multipath issues [11–13], addressed important aspects of the multipath routing. Cidon et al. and Rao and Batsell [11,12] discuss the reduction of the reservation delay and synchronization problems when their parallel multipath routing scheme is used, and Nelakuditi and Zhang [13] discusses methodologies of selecting the number of multiple paths in the context of the proportional routing (i.e. proportioned traffic over multiple paths).

In this section, we propose an enhanced routing algorithm stemming from the one in Ref. [8]. The originally proposed approach was to compute multiple QoS paths which may be overlapped with each other, but the number of links overlapped is minimized. Besides, the algorithm intelligently maintains network bridges[1] during the path-searching phase, and this leads to producing more paths. This source-based path computation approach becomes more practical in conjunction with the fast switching and path pinning-down method of MPLS. The newly proposed algorithm reduces the algorithm time complexity by running a single iteration of the algorithm and even finding more alternate paths. The new algorithm brings practical benefits of fault tolerance and load balancing in a cost-effective way. We evaluate the new algorithm by comparing it to the original algorithm. We also review a very fundamental multiple path algorithm which searches for fully disjoint multiple paths and motivated us to develop the first algorithm in Ref. [8]. We named them Fully Disjoint Multipath Algorithm (FDMA), Partially Disjoint Multipath Algorithm with Multiple Iteration (PDMA-MI), and Partially Disjoint Multipath Algorithm with Single Iteration (PDMA-SI). The primary purpose of this algorithm enhancement is to provide QoS path computation with more practicality in terms of path computation latency and resource requirement.

First, we briefly review the original algorithm in Ref. [8] (i.e. PDMA-MI) and the basic multiple QoS path approach (i.e. FDMA). After the review, we introduce the newly developed algorithm. Regardless of the algorithm differences, they all meet the following conditions.

- Satisfying given multiple QoS constraints.
- Minimizing hop count (i.e. minimizing network resources).
- Maximally disjoint (i.e. the number of overlapped links is minimized).

All paths searched by the algorithms must satisfy given multiple constraints such as bandwidth, delay, jitter, etc. This is because we assume that QoS-sensitive applications have a set of sophisticated constraints which must be all satisfied by the network. In addition, when searching for paths, the network resources should not be wasted by unnecessarily long paths. Therefore, the algorithms are required to minimize hop count. All the algorithms in this paper stem from the breadth-first search algorithm. Assuming that the current network condition is available at each node, traffic source examines its link state database by increasing hop count (i.e. breadth-first search). Thus, the time complexity of the algorithms becomes $O(V+E)$ for a network $G(V,E)$ where V and E are the numbers of nodes and links, respectively [14]. Since the gist of provisioning multiple QoS paths is to provide fault tolerance still supporting QoS guarantees, the algorithms must be able to find possibly many alternate paths. In addition, the searched

[1] Links separating its network into two isolated ones if it is removed from the network.

paths are expected to be apart from each other. This is not only because the more alternate paths are provisioned, the more reliable QoS services become with smaller possibility that link/node failures affect all paths, but also because more disjoint paths are insusceptible to the entire paths corruption due to network failures.

Although taking multiple constraints into account plays the core role, manipulating multiple constraints with QoS metrics optimization limits the actual number of QoS constraints to be dealt with and even becomes intractable problems. Thus, we designed the routing algorithms to relax the optimality issue and search for paths which satisfy all given QoS constraints regardless of how sufficiently they are satisfied. The definitions[2] in the following section are to support this essential rule with a breadth-first search manner.

2.1. Dealing with multiple QoS constraints

Definition 1. *QoS metrics*. Consider a network represented by a directed graph $G = (V, E)$ where V is the set of nodes numbered from 1 to N and E is the set of links (i.e. links $E(i,j)$ when $i \in V, j \in V, i \neq j$). All the links are assumed to be associated with R multiple QoS metrics. Each QoS metric is manipulated by its corresponding concatenation function. We generalize and compound the concatenation functions such that we have $F = \{F_1, \ldots, F_R\}$ for the R QoS metrics.

Definition 2. *QoS descriptor*. Along with the individual QoS metrics, QoS descriptor $D(i,j)$ is defined as a set of multiple QoS metrics associated with link $E(i,j)$; $\times$ $D(i,j) = \{q_1(i,j), \ldots, q_R(i,j)\}$ where $q(i,j)$ is the individual QoS metrics for $E(i,j)$. With $D_{qk}(i,j)$ defined as kth QoS metric in $D(i,j)$, $D(s,j)$ becomes $\{F_k(D_{qk}(s,i), D_{qk}(i,j)) | 1 \leq k \leq R\}$.

For simplicity, $D(i)$ means the QoS descriptor from the source to node i. Neighbors of node i are expanded with the QoS metrics associated with link $E(i,j)$ and $D(j)$ becomes $F(D(i), D(i,j))$ where $j \in N(i)$ and $N(i)$ is the set of neighbors of i. When the QoS metrics are iteratively concatenated, the QoS descriptors of nodes are evaluated if they satisfy given QoS constraints, and non-satisfying nodes are pruned

Definition 3. *Constraint verification*. A multiple QoS constraint set C is defined in the same format of D; $C = \{c_1, \ldots, c_R\}$ where each c defines individual QoS constraint. Each QoS metric in D is verified with corresponding constraints in C. A Boolean function $f_C(D)$ is defined to verify if D satisfies C; $f_C(D) =$ true if $c_k \in C, q_k \in D, c_k$ is satisfied by q_k for all $1 \leq k \leq R$. This finds paths of sufficient resources satisfying all the multiple constraints.

[2] The original works can be found in Ref. [8] and the notations have been revised for the extended algorithm.

The individual Boolean operations of c_k and q_k depend on their properties. We leave the specific characteristics of individual metrics abstract and assume that corresponding operators are properly defined. This allows us to focus only on verifying if QoS descriptors satisfy QoS constraint sets. When $f_C(D(i)) = true$, node i is *projected* and its neighbors are further expanded until destination d is reached. These properties are applied to all the algorithms in this paper.

2.2. Fully disjoint multiple QoS path algorithm (FDMA)

We first discuss an intuitive algorithm. We design this base algorithm such that it runs M times of the breadth-first search algorithm to find M paths. At each iteration of the search process, if a path is found successfully, all the links belonging to the path are removed from G. This leads the subsequent iterations to producing next alternate paths. This is a straightforward approach, but inherently it only produces fully disjoint paths.

In order to track where projected nodes pass, an extra field p is added in the QoS descriptor to record the preceding node from which the node of descriptor is reached; D becomes $\{q_1, \ldots, q_R, p\}$. To find the complete path, we follow the pointers p in the descriptors backward from the destination to the source after a path is successfully found. F' is a new generic concatenation function performing the same operations of and an additional operation on D_p. That is, when a qualified node j reached through i is projected, $D_p(j)$ becomes i.

In addition, the algorithms produce a set of paths as they proceed. Each path r is defined as a series of nodes whose sequence in the path represents the actual steps of the route. When the length of a path is l, the elements of the path include $r_1 \ldots r_l$ where r_k represents kth node in the path. Also, each path includes the QoS descriptor presenting the concatenated QoS metrics up to the last node of the path. The QoS descriptor of path r is denoted as r_D. With these additional definitions, Fig. 1 describes the details of FDMA. This simple approach is to be compared to PDMA-MI and PDMA-SI as the basic performance index.

2.3. Partially disjoint multiple QoS path algorithm with multiple iterations (PDMA-MI)

Like FDMA, PDMA-MI also finds M paths by running M times of the breadth-first search algorithm. However, PDMA-MI is not limited to fully disjoint paths. PDMA-MI examines how much the path being currently searched is overlapped with already searched paths. With this approach, links in the paths found in previous iterations are not eliminated. This increases the possibility of finding more alternate paths since bridges are always kept in the network.

For this approach, the QoS descriptor D becomes to include two new variables, n for 'new' and o for 'old,' to keep track of the degree of being disjoint; $D(i)$ becomes

Algorithm II.1: FDMA(s, d, n)

s : source node — d : destination node
n : maximum number of paths — G : graph (V, E) (i.e., topology)
S : path set — r : path
P, T, T' : QoS descriptor set — $N(x)$: neighbor nodes of node x
$D_p(i)$: preceding node of i

procedure COMPUTEPATH()
$P \leftarrow \emptyset$ $T \leftarrow D(s)$ from G // starts with the source node
// The loop ends when the destination is reached or no more expansion is possible.
while $D(d) \notin P$ **and** $T \neq \emptyset$
  $T' \leftarrow \emptyset$ // T' is to temporarily hold projected nodes
  for each $D(i) \in T$
    if $f_C(D(i)) =$ **true and** $D(i) \notin P$
      then $T' \leftarrow T' + D(i)$
      else discard $D(i)$
  NEXTHOP() // looks for neighbors of the projected nodes in T'
// If the destination is reached, the entire path is built by traversing D_p reversely.
build r from P
return (r)

procedure NEXTHOP()
// increases hop count by examining neighbors of the nodes in T'
// and the neighbors are recorded in T
$T \leftarrow \emptyset$
for each $D(i) \in T'$
  for each $D(j) \in N(i)$ from G
    if $j \neq D_p(i)$
      then $T \leftarrow T + F'(D(i), D(i, j))$

main
repeat
  $r \leftarrow$ COMPUTEPATH()
  if r is valid
    then $S \leftarrow S + r$
      // removes all the links in the found path from G
      // so that subsequent iteration finds fully disjoint path.
      remove all $E \in r$ from G
until r is invalid **or** $|S| = n$
return (S)

Fig. 1. Fully disjoint multipath algorithm (FDMA).

$\{q_1(i), \ldots, q_R(i), p, n, o\}$. These two variables are updated by checking if the node of the QoS descriptor has been already included in any previously computed paths. n increases when the node is not included in any previously computed paths, and o increases when it is detected in those paths. Fig. 2 shows the detailed algorithm.[3] In the algorithm, F'' is a variant of F' in FDMA. F'' performs the additional tasks on n and o as described.

2.4. Partially disjoint multiple QoS path algorithm with single iteration (PDMA-SI)

Now, we introduce PDMA-SI which is capable of finding multiple paths with just a single iteration of a breadth-first search algorithm. In PDMA-MI, each iteration projects qualified nodes by keeping track of the numbers of 'new' and 'old' links. This results that only one possible path to each qualified node becomes available by setting D_p to a better and single preceding node. In contrast, PDMA-SI keeps track of all possible paths in a single search phase. Instead of recording preceding nodes in the QoS descriptor of each node, PDMA-SI fully tracks the entire route of all feasible

[3] The original algorithm in Ref. [8] has been again revised and its details are reformatted in this paper.

Algorithm II.2: PDMA-MI(s, d, n)

s : source node — d : destination node
n : maximum number of paths — G : graph (V, E) (i.e., topology)
S : path set — r : path
P, T, T' : QoS descriptor set — $N(x)$: neighbor nodes of node x
$D_p(i)$: preceding node of i — $D_o(i)$: number of "old" links of i
$D_n(i)$: number of "new" links of i — $D'(i)$: projected QoS descriptor in P

procedure COMPUTEPATH()
$P \leftarrow \emptyset$
$T \leftarrow D(s)$ from G
// The loop ends only when there are no more nodes to be expanded.
while $T \neq \emptyset$
  $T' \leftarrow \emptyset$
  for each $D(i) \in T$
    if $f_C(D(i)) =$ **true**
      then
        if $D(i) \notin P$
          then $T' \leftarrow T' + D(i)$
          else if $D'_o(i) > D_o(i)$
          or $(D'_o(i) = D_o(i)$ **and** $D'_n(i) > D_n(i))$
        // Although a node has been already projected, if a more disjoint
        // path is found, D_p of the node is set to the better path.
          then $T' \leftarrow T' + D(i)$
          else discard $D(i)$
      else discard $D(i)$
  NEXTHOP()
build r from P
return (r)

procedure NEXTHOP()
$T \leftarrow \emptyset$
for each $D(i) \in T'$
  if $i = d$
    then skip i // Destination does not need to be expanded.
  for each $D(j) \in N(i)$ from G
    // When a neighbor is reached, QoS metrics are concatenated.
    // D_o and D_n are also updated by F''
    if $j \neq D_p(i)$
      then $T \leftarrow T + F''(D(i), D(i, j))$

main
repeat
  $r \leftarrow$ COMPUTEPATH()
  if r is valid
    then $S \leftarrow S + r$
      remove all $E \in r$ from G
until r is invalid **or** $|S| = n$
return (S)

Fig. 2. Partially disjoint multipath algorithm with multiple iterations (PDMA-MI).

paths as in the path vector algorithm of BGP [15]. Thus, if an intermediate node to the destination is qualified and reachable through two different preceding nodes, two paths are exclusively kept and further expanded separately. That is, a path *forks* when it is expanded at a node with more than one outgoing link. In this expansion process, the newly forked paths inherit the QoS descriptor of their original path and their individual QoS descriptors are verified with the links that they are expanded from. The original path is no longer kept and discarded from the search process. This path expansion is depicted in Fig. 3.

With respect to the number of searched paths, PDMA-MI has a limitation due to the nature of the search process. PDMA-MI does not fully discover all possible paths. An example is depicted in Fig. 4. Suppose that all the paths from r_1 through r_5 are feasible with all their links satisfying given multiple constraints. Paths r_4 and r_5 cannot be searched by PDMA-MI. After finding r_3, path (a, b, c) will be rejected in the subsequent search process (i.e. search

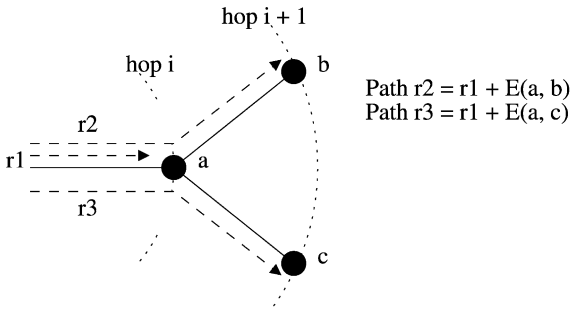

Fig. 3. Path expansion example in PDMA-SI.

process for r_4). This is because $E(a,b)$ and $E(a,c)$ are already used for r_2 and r_3 and they are considered as 'old' links. This prevents node c from being reached through node b. The same reason applies to the case of r_5.

Although PDMA-MI is capable of finding more alternate paths than FDMA by maintaining bridges, it is not able to fully discover whole possible cases. In contrast, PDMA-SI is capable of finding such ignored paths as well since all feasible paths are exclusively maintained and examined in the search process.

Although PDMA-SI is superior in finding more paths, the parallel search of all possible paths has a scalability issue. If a large network has a high degree of connectivity, the total number of paths concurrently searched by PDMA-SI becomes huge. This can be analyzed with complete graphs. All network topologies are subgraphs of a complete graph, and a complete graph of a given number of nodes has the largest number of possible paths between given two nodes. Thus, a complete graph of V nodes can have paths of hop count from 1 to $V-1$ since the paths must be loop-free. The total number of all possible paths is the sum of the number of paths of each hop. This can be formulated as follows.

$$P_V = \sum_{h=1}^{V-1} P_V(h) \tag{1}$$

$$P_V(h) = (V-2)P_{V-1}(h-1)$$

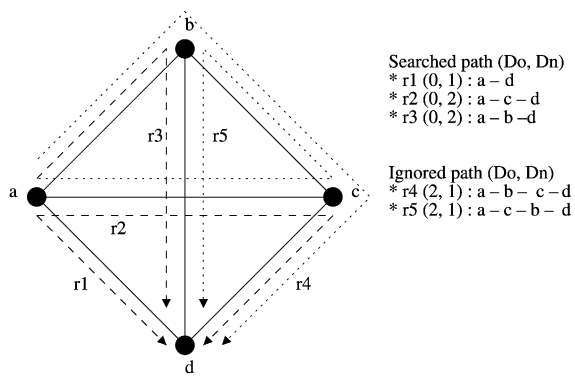

Fig. 4. Path search limitation of PDMA-MI.

$$\vdots$$

$$P_V(h) = (V-2)(V-3)\cdots(V-h)P_{V-h+1}(1)$$

$$P_V(h) = \frac{(V-2)!}{(V-h-1)!}$$

$$P_V = \sum_{h=1}^{V-1} \frac{(V-2)!}{(V-h-1)!}$$

$P_V(h)$ is the number of paths of hop count h and P_V is the entire sum of them. Eq. 1 shows that P_V exponentially increases as V grows. Thus, the parallel search may demand a huge memory space to hold many concurrently searched paths. However, we can remedy this by adding a condition to the search process. Basically, the parallel search process blindly deals with redundant and unnecessarily expanded paths. We define a path to be redundant if the path is expanded over a link whose opposite direction has been already used for path expansion in the search process (assuming all links are directed edges and treating each direction of links independently). That is, if a direction of a link has been explored for path expansion, the direction of the link is deemed to be *outbound* from the source since the algorithm expands paths by increasing hop count. Thus, if a path is expanded through the opposite direction of the link, the path is reaching a node through unnecessarily long route and it is considered to be an expansion through a *inbound* route. This condition is depicted in Fig. 5.

Fig. 5 shows an example of path expansions over a complete graph of 5 nodes (i.e. K_5 graph). When source node a searches for paths in parallel to destination node a by increasing hop count, the pictured expansions will occur. If the algorithm blindly searches, all the paths up to hop count 4 will be discovered. However, we designed the algorithm to prevent the *inbound* paths from being expanded. For instance, at hop count 2 in Fig. 5, $E(c,b)$ is excluded since its opposite direction $E(b,c)$ on the link has been already visited at hop count 1. The same reason is applied to $E(b,c), E(d,b)$, $E(b,d)$, $E(d,c)$, and $E(c,d)$. We call this condition 'Outbound-Path-Only (OPO).' We apply this condition at each time the algorithm increases hop count.

With this condition applied, we can reduce the huge number of alternate paths by preventing redundant paths from being generated. We executed the algorithm with OPO

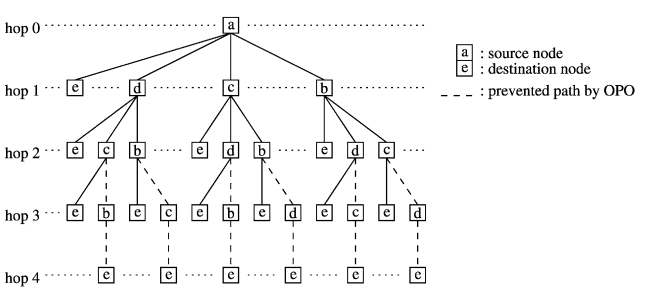

Fig. 5. Path expansion with the outbound-path-only (OPO) option.

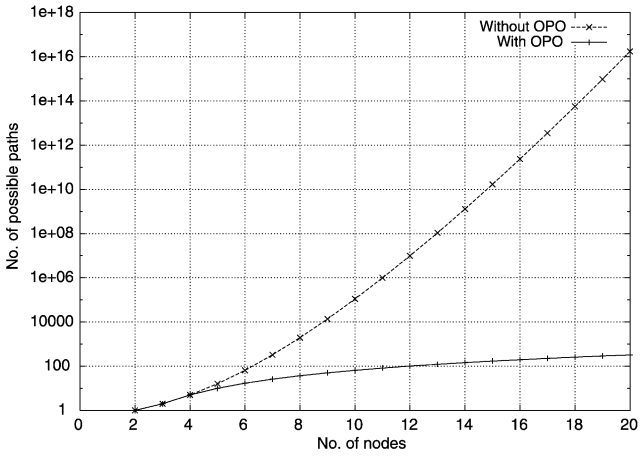

Fig. 6. Comparisons of the total number of alternate paths in complete graphs with or without outbound-path-only (OPO).

over complete graphs of different numbers of nodes to figure out how many alternate paths are searched. For the same graph, we also computed the total number of possible alternate paths with Eq. 1. Fig. 6 shows the result and the significant impact of OPO. This is the factor that really makes the algorithm practical.

In addition, we added a pre-search process making the network topology compact by removing stub nodes. Stub nodes unnecessarily allow the path expansion process to generate paths which are eventually not reaching destination at all. Thus, prior to the execution of the search algorithm, all stub nodes excluding source and destination are recursively eliminated.

While PDMA-MI searches for paths iteratively in the order of the defined preferences, PDMA-SI searches for alternate paths in parallel. Then, it sorts the found paths. The sorting conditions are the same as in PDMA-MI. However, if M paths are to be searched by PDMA-SI, the results will not be what PDMA-MI produces. This is because PDMA-SI collects paths in the order of reaching destination due to the breadth-first search nature. First M paths collected by PDMA-SI may not be as much disjoint as the ones produced by PDMA-MI. Thus, the path sorting process is performed after collecting sufficient paths. The sufficient number of paths is defined to be $V-1$. The rationale is that there are at least $V-1$ disjoint paths in a complete graph of V nodes, and further searched paths are inherently overlapped with some of previously found paths. Thus, PDMA-SI tries to find up to $V-1$ paths and they are sorted according to the same preferences defined in PDMA-MI. Then, a certain number of paths among the searched ones are selected for actual use with the path management schemes in Ref. [8]. Combining all these properties, Fig. 7 illustrates the PDMA-SI details.

As shown, the multiple path computation algorithms and relevant enhancements play the core role in the QoS testbed implementation. In the following section, the testbed system

Algorithm II.3: PDMA-SI(s, d, n)

s	: source node	d	: destination node
n	: maximum number of paths	G	: graph (V, E) (i.e., topology)
S, T, T'	: path set	r, r'	: path
L	: edge set		

procedure NEXTHOP(r)
// When neighbors are projected, the expansion directions are recorded in L such that
// they are prevented in subsequent hop of the search process.
$T' \leftarrow \emptyset$
$L \leftarrow \emptyset$
for each $j \in N(r_l)$
  if $f_C(F(r_D, D(r_l, j))) =$ **true**
  then
    $r' \leftarrow r \cdot j$ // concatenates r with the neighbor j
    $r'_D \leftarrow F(r_D, D(r_l, j))$
    $T' \leftarrow T' + r'$
    $L \leftarrow L + E(r_l, j)$

procedure SORTPATH()
// On the completion of the search process, paths are recorded in the order of being found.
// The paths are sorted according to the defined preferences.
$S \leftarrow \emptyset$ // is to hold the sorted paths.
while $|S'| > 0$
  // Each iteration picks the best path and stores it in S
  for each $r \in S'$
    // All paths in S' are compared to the ones in S and their
    // D_o and D_n are updated.
    update r_{D_o}, r_{D_n} by comparing r to all $r' \in S$
  sort all $r \in S'$
  // sorting condition: r is more preferable than r' when
  // $(r_{D_o} < r'_{D_o})$ **or** $(r_{D_o} = r'_{D_o}$ **and** $r_{D_n} < r'_{D_n})$
  pick the best r from S'
  $S \leftarrow S + r$
  $S' \leftarrow S' - r$

main
PRUNESTUBNODE() // recursively removes all stub nodes.
$S' \leftarrow \emptyset$
$r_1 \leftarrow s$
$T \leftarrow r$
while $|T| > 0$ **and** $|S'| < n$
  $L \leftarrow \emptyset$
  for each $r \in T$
    if $r_l = d$
      then $S' \leftarrow S' + r$
      else NEXTHOP(r)
  remove all $E \in L$ from G // performs the OPO option.
  $T \leftarrow T'$
SORTPATH()
return (S)

Fig. 7. Partially disjoint multipath algorithm with single iterations (PDMA-SI).

architecture is introduced with various network components.

3. System architecture

The testbed consists of PCs running Linux, and all the QoS-capable features are embedded in the Linux kernel. Each of the machines has several modules running on it, namely the link emulator, metric collector, OSPF daemon, MPLS forwarding and the applications, and the entire system architecture is depicted in Fig. 8. The following sections describe in more detail each of these modules individually, explaining their functions and their implementation issues.

3.1. Link emulator

The testbed that we implement uses the wired Ethernet LAN in the lab, as the physical communication medium. Since we want to emulate several scenarios with widely

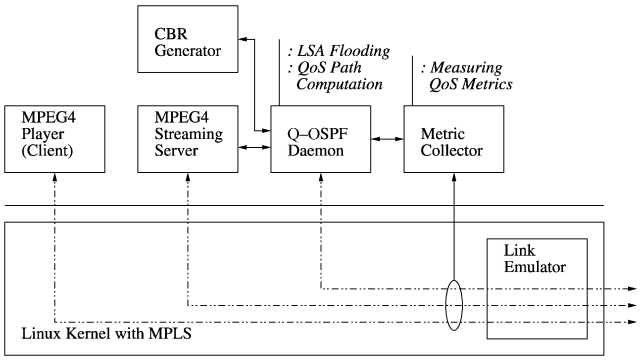

Fig. 8. The entire QoS system architecture for the experiment testbed.

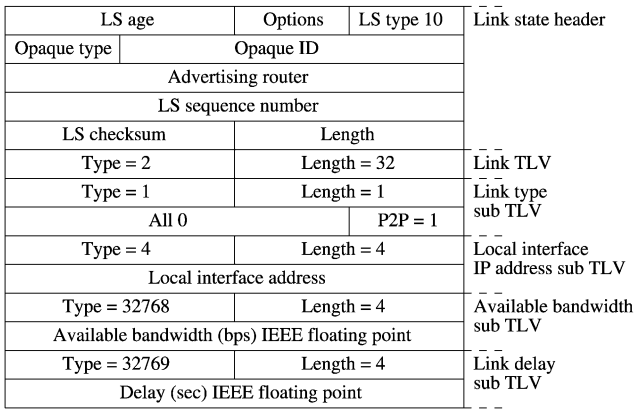

Fig. 9. Opaque LSA format.

varying link capacities and propagation delays, we require a network emulator to do the job for us. We emulate the characteristics of bandwidth and delay over a link using `tc`, the Linux kernel traffic shaper and controller. `tc` can emulate a wide variety of policies with hierarchies of filters and class based queues on the outgoing interfaces. We require simple bandwidth and delay emulation which can be done quite easily.

A simple '`tc qdisc add dev eth0 root tbf rate 10 kbps latency 10 ms`' emulates a 10 Kbps link with a latency of 10 ms on the `eth0` interface. Since we have bursty multimedia traffic, the buffer size is also an important factor, and after a few experiments, we settled on a value of 25 Kbits as a reasonable buffer to accommodate burstiness.

3.2. Metric collector

To provide QoS, reserving bandwidth and bounding on delay for a connection, we require the knowledge of link characteristics at all times. This information is needed by the QoS allocation module which looks at the current usage of the network and figures out if the new request can be satisfied or not. The link metric collection module, therefore, is an integral part of any routing scheme implementation that provides QoS routing.

The OSPF daemon (OSPFD) implementation does not have any link characteristics measurement module. The metrics we are interested in are the available bandwidth in the link and the delay. One could think of other metrics also, such as queue length, loss probability, etc. but bandwidth and delay are the two most important metrics used in QoS routing. Thus, our design goal was to write a module to measure these two metrics and integrate this code with the existing OSPFD implementation.

For bandwidth metric collection, we opted to simply use the log file maintained in `/proc/net/dev/` which contains information about the number of packets and bytes received and transmitted on each interface. By examining this file at regular intervals, we calculate the bandwidth used on the each outgoing interface. Delay metric collection is done using the 'ping' utility to send ping probes to the other side of the link and collect the delay value. The metric collection code sends ping message and collects bandwidth values from the `/proc/net/dev` log file, every time interval (typically 10 ms). The values collected are exponentially averaged to smooth out the fluctuations.

3.3. Q-OSPF daemon

To propagate QoS metrics among all routers in the domain, we need to use an Interior Gateway Protocol (IGP). OSPF is one of the major IGPs and significant researches have been recently made on OSPF with traffic engineering extensions. We selected the open source OSPFD [10,16] to implement our QoS routing scheme. [17] defines Opaque LSA (Link State Advertisement) for OSPF nodes to distribute user-specific information. Likewise, we define our specific Opaque LSA entries by assigning new type values in the Opaque LSA format shown in Fig. 9.

When OSPFD runs at routers, it tries to find its neighbor nodes by sending HELLO messages. After establishing neighbor relationship, OSPFD asks the metric measurement module to calculate the QoS metrics of the established link. OSPFD gives as input the local interface address and the neighbor router interface address to the metric measurement module and generates the opaque LSA for each interface. The LSA contains the available bandwidth and queuing delay metric obtained from the metric measurement module. LSA update frequency is currently restricted to MinLSInterval (5 s). In our implementation, we followed this restriction, but we feel that the LSA update interval is an important parameter for the correctness of our algorithms because if there are too frequent changes to the generated LSA, there will be miscalculations during the QoS path computation due to the flooding delay.

In addition to LSA flooding, OSPFD exchanges router LSAs to build a full network topology. Router LSAs originate at each router and contain information about all router links such as interface addresses and neighbor addresses. We bind the link metrics that we are interested

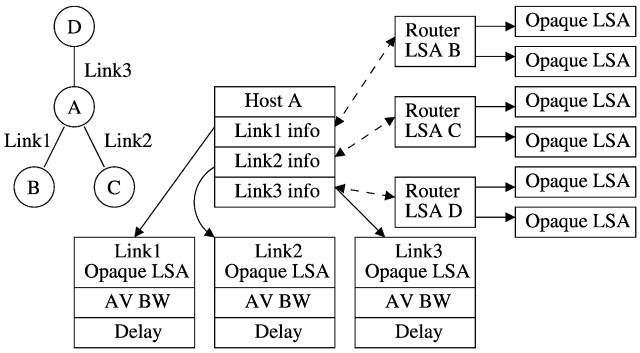

Fig. 10. Router LSAs.

in, viz. bandwidth and delay to the opaque LSA specified by the link interface address. Thus, we introduce the pointer from each link of a router LSA to the corresponding opaque LSA as shown in Fig. 10. Whenever OSPFD receives router LSAs or opaque LSAs, it just updates the link pointer to point to the new traffic metrics reported by that link.

3.4. MPLS

One of the key assumptions for Q-OSPF to be effective is the capability of setting up explicit paths for all packets of a stream to use. Thus, we need to pin down the path that a connection uses. One of the main advantages of MPLS is its efficient support of explicit routing through the use of Label Switched Paths (LSPs). With destination-based forwarding as in the conventional datagram networks, explicit routing is usually provided by attaching to each packet the network-layer address of each node along the explicit path. This approach makes the overhead in the packet prohibitively expensive. This was the primary motivation behind deploying MPLS at the lower layer instead of using something like IP source-routing or application level forwarding.

3.5. Applications

An integral part to the whole process of making a testbed was the development of applications running on top of this architecture. These applications request QoS-constrained routes, generate traffic, and in turn change the QoS metrics of the network. We used the open source *Darwin Streaming Server* distributed by Apple [18] to stream MPEG-4 files using RTSP over RTP. The open source `mp4player` from the MPEG4IP project [19] was used to play these files over the network. Modifications to the source code involved extending the capabilities of the server and the player for streaming and playing over multiple paths, and designing an interface for the server to interact with Q-OSPF daemon and request paths to the client. For the experiments involving CBR traffic, we used a home-brewed traffic generator which generates UDP traffic at a given rate in either single path or multiple path mode.

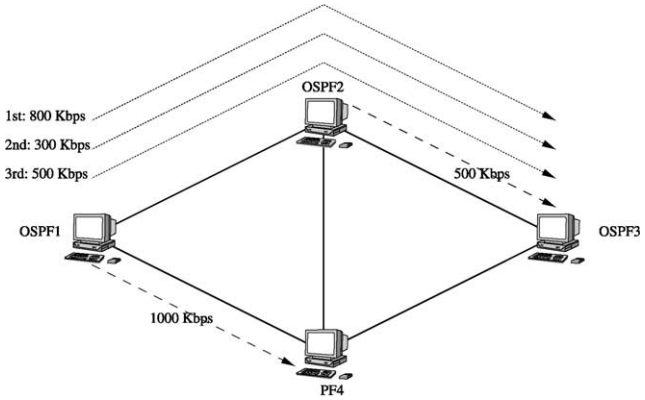

Fig. 11. The first experiment topology and corresponding network traffic.

4. Experiments

To evaluate the QoS network system implementation and also to validate the simulation results, we conducted two sets of experiments. The first set of experiments shows the QoS capability of the implemented system. Fig. 11 shows the network topology for the first set of experiments. 4 nodes (i.e. routers) are connected directly to each other through 1.5 Mbps links. The traffic injected into the testbed is Constant Bit Rate (CBR) traffic with 1000 byte UDP packets. The CBR traffic is intended to model real time video distribution or video conference applications both requiring QoS support. There are two extra connections creating interfering background traffic: a 1000 Kbps connection from OSPF1 to OSPF4 and a 500 Kbps connection from OSPF2 to OSPF3.

We introduce three new connections (800, 300, 500 Kbps) from OSPF1 to OSPF3 at staggered starting times. Firstly, we examine the performance of the conventional IP routing which does not provide any QoS capability. In this case, packets of all the connections are routed over the shortest path (OSPF1, OSPF2, OSPF3). Thus, all three connections share the same path and cause

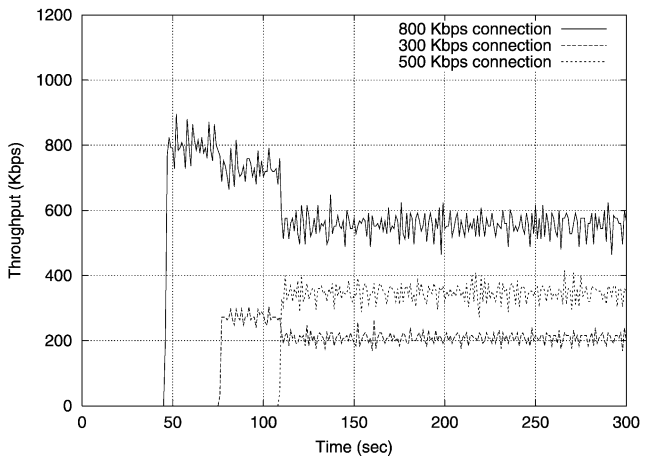

Fig. 12. Connection throughput measured from the system with the conventional IP routing (i.e. min-hop routing).

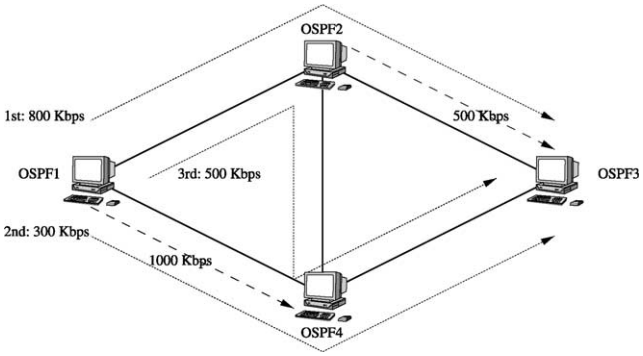

Fig. 13. Path selections for the given network connections with QoS routing.

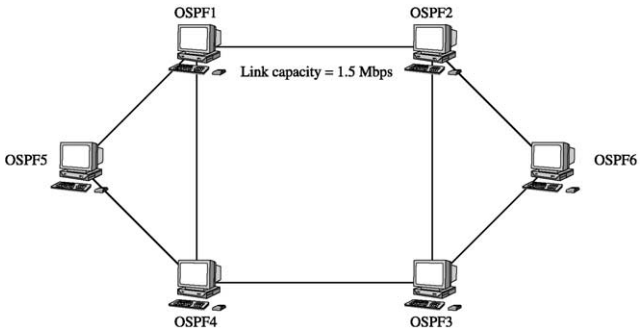

Fig. 15. The network topology for multiple QoS path fault tolerance: 6 nodes.

network congestion on the link from OSPF2 to OSPF3, while other links are totally empty and unused. This is an inevitable condition due to the lack of the QoS capability in the current IP networks.

Fig. 12 shows the throughput of the connections as a function of time when the conventional IP routing (i.e. min-hop routing) is used. It can be seen from the graph that the throughput are proportionally decreased after the connections are inserted and their traffic exceeds the link capacity between OSPF2 and OSPF3. The throughput of each connection eventually drops to around 70% of their desired bandwidth. This is an expected result and we can analyze this from the experiment parameters; link capacity over the incoming traffic rate (i.e. 1.5 Mbps/(500 + 800 + 300 + 500 Kbps)) is equal to 0.71.

In contrast, the enabled QoS routing feature in the system shows the capability of routing the given connections over the paths with sufficient resources. Fig. 13 depicts the path selections for the given connections with the QoS routing.

The path computation process in the extended OSPF daemon of the system computes three different paths as expected for the incoming connections. Fig. 14 shows the throughput of the connections when the QoS routing feature becomes effective. All the connections completely meet their requirements by avoiding congestion. This experiment proves that appropriate paths are selected by the QoS routing capability with given QoS constraints (e.g. bandwidth constraints).

The second set of experiments examines the fault tolerance capability by provisioning multiple QoS paths between source and destination and spreading packets over the multiple paths. Figs. 15 and 16 show the network topology for the second experiments. In the first instance, six network nodes (emulated by PCs) were connected to each other through 1.5 Mbps links. We inserted 30 connections generating 100 Kbps CBR traffic each. The source and destination pairs are randomly selected. Random link failures are 'forced' during the execution of the experiments. For the link failure model, the state transition of each link from the link-down state to link-up is geometrically distributed with 5 s. Likewise, the transition from link-up to link-down is also

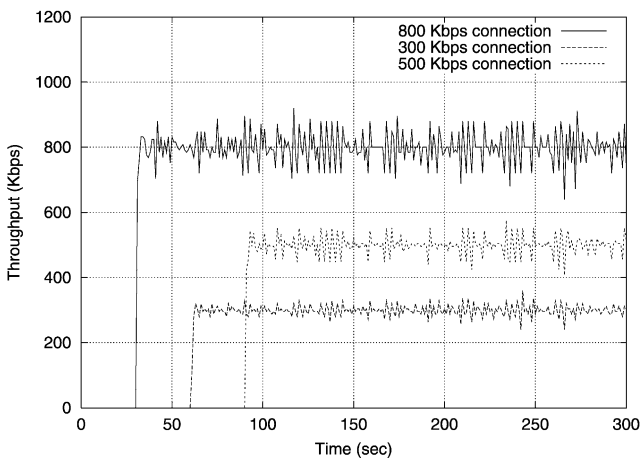

Fig. 14. Connection throughput measured from the system with QoS routing.

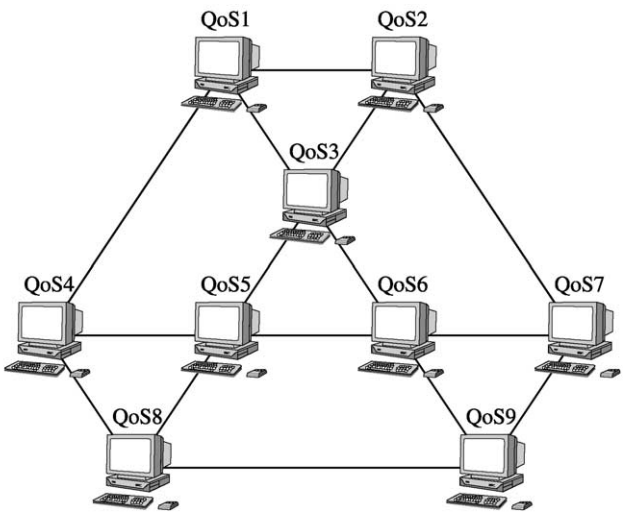

Fig. 16. The network topology for multiple QoS path fault tolerance: 9 nodes.

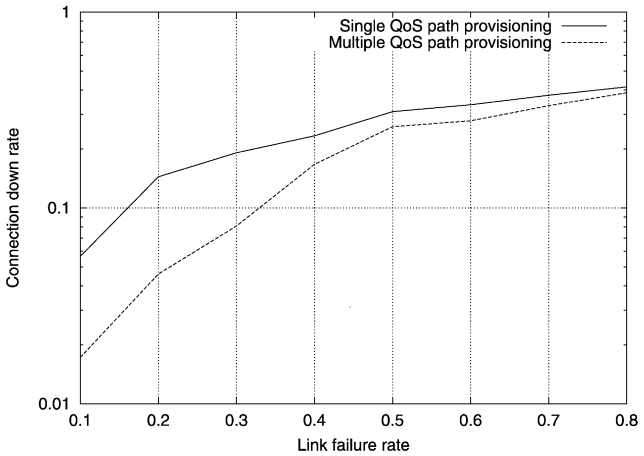

Fig. 17. Connection down rates as a function of link failure rates: 6 nodes.

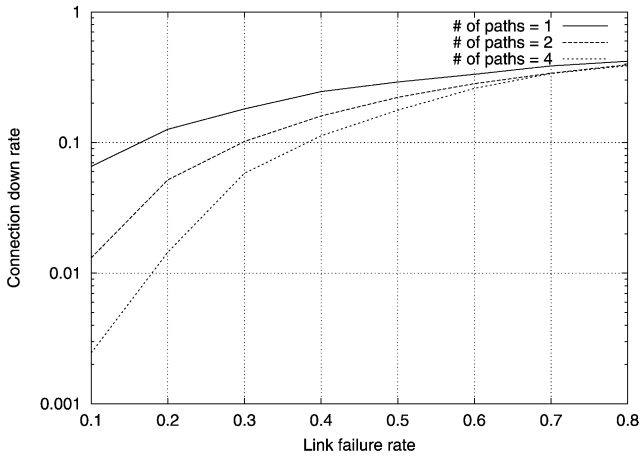

Fig. 18. Connection down rates as a function of link failure rates: 9 nodes.

geometrically distributed. The mean down time is easily derived from the link-down rate. In the second instance, we have the same characteristics, except that we have nine nodes and 100 connections. We also control the maximum number of multiple paths that a connection can use since that is expected to be an important implementation factor. In the six node case, we did not have any choice, since only two path options existed. In this case, we have up to four multiple paths, so we experimented for MAX-MULTIPLE-PATH for 1 (single path case), 2, and 4.

The performance statistic collect during this experiment is the down time for each connection. When a link on the path goes down, the packets on the connection stop going through. They may, however, flow on a back up path if one were available. When the link comes up, the connection resumes transmission on that link. A connection is defined to be 'down' when none of its packets reach destination (i.e. all paths have failed). The duration of the experiments was 10 min. Figs. 17 and 18 show the connection down-time percentage when single path or multiple paths are provisioned for the connections. The result shows that multiple path provisioning has lower connection down rate[4] as expected. Note that these experiments are slightly different from the simulation experiments reported in Ref. [8]. In Ref. [8], the authors assume that a connection gets aborted when all its paths have one or more faults, and does not come up automatically when the faults get repaired. Note that the difference between the multi-path fault tolerance and the single-path fault tolerance is greater in the 9 node scenario compared to the 6 node scenario. This is to be expected, since in the 9 node topology, there is much greater redundancy, resulting in more multi-paths to be used (around 4) compared to the 6 node case (around 2). In fact, as Fig. 18 shows, the results are similar for 6 nodes and 9 nodes if we constrain the 9 node topology to use just 2 multiple paths. Thus the more multiple paths are available, the better the fault tolerance, so these benefits are expected to even greater for larger networks.

[4] Connection down rate = connection down time/connection duration.

5. Conclusion

We presented a QoS testbed with multiple QoS path computation algorithms. The recent algorithm enhancement (PDMA-SI) showed its practicality in terms of path computation latency, resource consumption, and the number of searched paths. Based on the multiple path provisioning, the testbed has been implemented on Linux-based PCs to provide an emulation environment for design and evaluation of QoS-related protocols and algorithms. We have evaluated various aspects of the Q-OSPF algorithm including throughput satisfaction and fault tolerance support as described in Ref. [8]. We have experimented with various topologies ranging from four nodes to nine nodes. The testbed results closely match simulation. They prove that these algorithms can actually be implemented and deployed in practice, yielding attractive benefits following the use of techniques like multi-path packet scattering.

The emulation environment as described in the paper was in a very active stage of evolution. Since then, we have added MPLS support in the kernel for explicit path routing. We have also developed and simulated a scheme for fast provisioning of QoS paths in a hierarchical network and plan to implement it on the testbed. Current goals involve making the flow-identification scheme implicit by putting in packet classifiers at the edge routers, which will provision routes without the application layer being aware of it. An orthogonal direction is the creation of a QoS-aware socket system call which will allow the application to simply open a socket with the QoS constraints specified, and start sending on it, while the route provisioning and calculation

happens in the lower layers. We are working in both these directions at the moment.

References

[1] R. Guerin, A. Orda, D. Williams, QoS routing mechanisms and OSPF extensions, Proceedings of Global Internet (Globecom), Phoenix, Arizona November (1997).

[2] D. Cavendish, M. Gerla, Internet QoS routing using the Bellman–Ford algorithm, IFIP Conference on High Performance Networking (1998).

[3] G. Apostolopoulos, S. Kama, D. Williams, R. Guerin, A. Orda, T. Przygienda, QS routing mechanisms and OSPF extensions, request for comments 2676, Internet Engineering Task Force August (1999).

[4] A. Dubrovsky, M. Gerla, S.S. Lee, D. Cavendish, Internet QoS routing with IP telephony and TCP traffic, Proceedings of the ICC June (2000).

[5] E. Rosen, A. Viswanathan, R. Callon, Multiprotocol label switching architecture, request for comments 3031, Internet Engineering Task Force January (2001).

[6] S. Chen, K. Nahrstedt, An overview of quality of service routing for next-generation high-speed networks: problems and solutions, IEEE Networks 12 (6) (1998) 64–79.

[7] H. Schulzrinne, Keynote: Quality of Service—20 Years Old and Ready to Get a Job?, Lecture Notes in Computer Science, vol. 2092, International Workshop on Quality of Service, June 2001, pp. 1.

[8] S.S. Lee, M. Gerla, Fault tolerance and load balancing in QoS provisioning with multiple MPLS paths, Lecture Notes in Computer Science, vol. 2092, International Workshop on Quality of Service (IWQoS), 2001, pp. 155.

[9] S.S. Lee, G. Pau, Hierarchical approach for low cost and fast QoS provisioning, Proceedings of IEEE Global Communications Conference (GLOBECOM) November (2001).

[10] J. Moy, OSPF Version 2, Request for Comments 2328, Internet Engineering Task Force April (1998).

[11] I. Cidon, R. Rom, Y. Shavitt, Multi-path routing combined with resource reservation, Kobe, Japan April (1997) 92–100.

[12] N.S.V. Rao, S.G. Batsell, QoS routing via multiple paths using bandwidth reservation, San Francisco, California March/April (1998) 11.

[13] S. Nelakuditi, Z.-L. Zhang, On selection of paths for multipath routing, IWQoS (2001) 170–186.

[14] H.C. Thomas, E.L. Charles, L.R. Ronald, Introduction to Algorithms, McGraw-Hill, New York, 1990.

[15] Y. Rekhter, T. Li, A border gateway protocol 4 (BGP-4), request for comments 1771, Internet Engineering Task Force March (1995).

[16] J.T. Moy, OSPFD Routing Software Resources, http://www.ospf.org.

[17] R. Coltun, The OSPF Opaque LSA Option, Tech. Rep.

[18] Apple Computer, The Darwin Streaming Server, http://www.opensource.apple.com/projects/streaming.

[19] B. May D. Mackie, A.M. Franquet, The MPEG4-IP Project, http://mpeg4ip.sourceforge.net.