# Hierarchical Approach for Low Cost and Fast QoS Provisioning

Scott Seongwook Lee, Giovanni Pau
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095-1596

***Abstract*- In order to provide practical QoS services, we propose a new traffic engineering approach for low-cost and fast QoS provisioning. The new scheme utilizes network resources more evenly by exercising an enhanced QoS path computation algorithm that finds maximally disjoint multiple QoS paths. An efficient mechanism for managing computed paths and allocating calls is also discussed as a component of the system, and this path management allows fast QoS provisioning by avoiding unnecessary path recomputation. Moreover, the naive approaches to link state information acquisition in conventional QoS routing are replaced with a more effective approach dividing the link state update process into three hierarchical subprocesses. Via simulation experiments based on IP Telephony application, we show that we can improve the performance and at the same time reduce the link overhead.**

## I. Introduction

For recent years, many substantial research efforts for QoS support in IP networks have been made [1], [2], [3]. [1] addresses practical QoS extensions in OSPF considering bandwidth and delay constraints to provide QoS support in intradomain. [2] more precisely analyzes QoS routing approaches optimizing network resources and minimizing hop count by taking advantage of the Bellman-Ford algorithm properties, and [3] proves the efficiency of the proposed QoS routing and call admission control (CAC) for given IP Telephony traffic. This fundamental groundwork shows promising future of QoS-enabled IP networks with the approaches in the traffic engineering.

Likewise, we propose a new approach in the traffic engineering to support QoS services in IP networks in more cost-effective and less time-consuming fashion. The new approach is based on the measurement-based QoS routing and assumes that MPLS [4] runs for the packet-forwarding method. Path computations and routing decisions in this approach are made at edge routers and paths are pinned down by MPLS as in [3]. Compared to the QoS routing and CAC schemes in the previous research works, our new scheme must be cost-effective with respect to consuming network resources to collect up-to-date link state information for QoS routing computation since hierarchical separation of link state information acquisition is carried out, while the conventional approaches in QoS routing mainly depend on frequent flooding to exchange link state information between nodes. Thus, this new scheme is named Hierarchical Approach for QoS Provisioning (HAQP). Moreover, HAQP is equipped with an enhanced QoS routing computation algorithm which finds maximally disjoint multiple QoS paths. We recall that the conventional QoS routing and CAC schemes compute single QoS paths to admit incoming calls and the outcomes of the computations remain the same until the previously computed path is fully saturated. Thus, possible paths from a source to a destination are consumed one-by-one. On the other hand, making parallel use of multiple paths utilizes network resource more evenly and focussed congestion can be avoided in early stage. It also fundamentally soothes the need of frequent refreshment of link state database since the impact of dynamic changes of admitted calls is diffused over multiple paths. In conjunction with the multiple path computation, an efficient path management scheme is embedded for fast QoS provisioning with unnecessary path recomputation avoided.

## II. System Architecture

The fundamental task of QoS routing is to find paths satisfying given QoS constraints and even considering the optimization of resource utilization which makes approaches to QoS routing difficult. In addition, the dynamic changes of network state make it expensive to gather up-to-date information in a large network. Recalling the premises for the primary goal of the new QoS provisioning approach, we designed the system with three distinct components; making use of multiple QoS paths, path management with quantized QoS constraints, and hierarchical link state information acquisition. Throughout this section, these three components are discussed in turn.

### A. Multiple QoS Path Computation

The primary purpose of computing and making use of multiple QoS paths in a parallel manner is to broaden the region of utilized resources and slow down the saturation of resource utilization. When multiple possible paths are available, the entire resource consumption of each path can be delayed and the dynamic changes of offered traffic may not cause severe load fluctuation in each path. In order to search for multiple paths, we define alternate paths with the conditions; satisfying given multiple QoS constraints, *maximally disjoint* from already computed paths, and minimizing hop count. Our proposed algorithm is a heuristic solution, and we do not limit ourselves to strictly "path disjoint" solutions. Rather, we search for multiple, maximally disjoint paths such that the number of overlapped links for different paths becomes as minimal as possible. This multiple QoS path computation algorithm produces incrementally a single path at each iteration rather than multiple paths at once. All the previously generated paths are kept into account in the next path computation. The efficiency of this algorithm for fault tolerance and load balancing is shown in [5].

*Definition 1:* QoS metrics. Consider a network represented by a graph $G = (V, E)$ where $V$ is the set of nodes and $E$ is the set of links. Each link $(i, j) \in E$ is assumed to be associated with $R$ multiple QoS metrics. These QoS metrics are categorized mainly into *additive*, *transitive*, and *multiplicative* ones. Each QoS metric is manipulated by their corresponding concatenation functions, and regardless of the specific properties of the functions, we generalize and compound the functions such that we have $F = \{F_1, \ldots, F_R\}$ since $R$ multiple QoS metrics are assumed to be associated with each link.

*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 $(i, j)$; $D(i, j) = \{q_1(i, j), \ldots, q_R(i, j)\}$. With $D_{q_l}(i, j)$ defined as $l^{th}$ QoS metric in $D(i, j)$, $D(1, j)$ becomes $\{F_l(D_{q_l}(1, i), D_{q_l}(i, j)) | 1 \leq l \leq R\}$.

The nodes of $G$ are numbered from 1 to $n$, so $N = \{1, 2, \ldots, n\}$. We suppose without loss of generality that node

1 is the source. $D(i)$ implies the QoS descriptor from the source to node $i$. Neighbors of node $i$ are expanded with the QoS metrics associated with link $(i, j)$ and $D(j)$ becomes $F(D(i), D(i, j))$ where $j \in N(i)$ and $N(i)$ is the set of neighbors of $i$. The algorithm iteratively searches for the QoS metrics of all reachable nodes from the source as the hop count increases. In this procedure, the QoS descriptors of the nodes must be checked if they satisfy given QoS constraints. If not satisfying, the nodes of the non-satisfying descriptors are *pruned* to search only the constraint-satisfying paths.

*Definition 3:* Constraint verification. A set of multiple QoS constraints is defined as $Q$ and the set is in the same format of $D$; $Q = \{c_1, \ldots, c_R\}$ such that each QoS metric in $D$ is verified with corresponding constraints. A Boolean function $f_Q(D)$ is also defined to verify if $D$ *satisfies* $Q$. The verification is to find paths of sufficient resources satisfying all the multiple constraints without considering how sufficiently the individual metrics satisfy the constraints. $f_Q(D) = 1$ if $c_l \in Q, q_l \in D, c_l$ is satisfied by $q_l$ for all $1 \leq l \leq R$. Otherwise, $f_Q(D) = 0$.

We leave the specific characteristics of individual metrics abstract and assume that they are properly verified by corresponding operators. Now, we focus only on verifying if the QoS descriptors of each node satisfy the QoS constraints, rather than verifying whether or not the individual QoS metrics are qualified since the QoS metrics all together must lie in the feasible region of the given QoS constraints. When $f_Q(D(i)) = 0$, node $i$ gets *pruned* by the algorithm. Otherwise, it is *projected* and its neighbors are further expanded.

*Definition 4:* New or old paths. A new field $p$ is added to the QoS descriptor to determine where the projected nodes pass. Besides, the QoS descriptor is augmented with two new variables, $n$ for "new" and $o$ for "old," to keep track of the degree of being disjoint. 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. These two variables play the most important role in the multiple path computation algorithm such that a path *maximally disjoint* from previously computed paths can be found. $D(i)$ becomes $\{q_1(i), \ldots, q_R(i), p, n, o\}$.

*Definition 5:* The multiple path computation algorithm.

$T = \{D(1)\}$
$P = \emptyset$
$h = 0$
**while** $T \neq \emptyset$ **do**
Step 0: $T' = \emptyset$
**for each** $D(i) \in T$ **do**
**if** $f_Q(D(i)) = 1$ **then** $T' \leftarrow D(i)$
Step 1: **for each** $D(i) \in T'$ **do**
**if** $D(i) \notin P$ **then** $P \leftarrow D(i)$
**else if** $D'_o(i) > D_o(i)$
**or** $(D'_o(i) = D_o(i)$ **and** $D'_n(i) > D_n(i))$ **then** $P \leftarrow D(i)$
**else discard** $D(i)$ **from** $T$
Step 2: $T = \emptyset$
**for each** $D(i) \in T'$ **do**
**if** $i = d$ **then skip** $i$
**for each** $j \in N(i)$ **do**
**if** $j \neq D_p(i)$ **then** $T \leftarrow F'(D(i), D(i, j))$
**else skip** $j$
$h = h + 1$

where $T$ is a temporary set of QoS descriptors and initialized with the QoS descriptor of source node 1, and $P$ is the set which collects all the qualified QoS descriptors of projected nodes. $D'(i)$ in Step 1 means the QoS descriptor of node $i$ which has been already projected and included in $P$. The algorithm does not stop just after finding the destination but instead keeps investigating all qualified nodes. Since the algorithm does not end on finding $d$ and instead it ends after finding all qualified nodes (i.e., $T = \emptyset$), the neighbors of $d$ are kept from being reached through $d$ such that unnecessary routes going through $d$ are not constructed. The iteration runs over the entire nodes until no further expansion can occur and $P$ becomes to include all qualified nodes. As this comprehensive iteration runs, newer routes to each node are constructed by Step 1. $F'$ is a new function based on $F$, and it performs the concatenation and records the preceding node in $D_p$. In addition, $F'$ updates the new two variables in the QoS descriptor, $n$ and $o$, as discussed in Definition 4. $F'$ checks if the link between $i$ and its neighbor $j$ has been already included in any previously computed paths. If the link is found in the previously computed paths, $D(j)$ expanded from $D(i)$ through the link $(i, j)$ becomes to have $D_o(i) + 1$ for its $D_o(j)$. Otherwise, $D_n(j)$ becomes $D_n(i) + 1$.

Now that $o$ and $n$ are updated in the described way whenever neighbors of nodes are discovered, the sum of their values represents the hop counts to the nodes from 1. The values of $o$ and $n$ are used in Step 1 which determines which route is newer and more preferable in terms of hop counts. Although the algorithm prefers newer paths compared to previously computed paths, it does not mean that the algorithm finds longer paths. Instead, it looks for a path more disjoint from previously computed paths, but if several new paths exist and their degree of being disjoint are the same, the shortest one among them is selected. This is achieved by the condition in Step 1, **if** $D'_o(i) > D_o(i)$ **or** $(D'_o(i) = D_o(i)$ **and** $D'_n(i) > D_n(i))$ **then** $P \leftarrow D(i)$.

This multiple QoS path computation algorithm searches for maximally disjoint paths through the nodes satisfying given QoS constraints yet minimizing hop counts. This satisfies our premise for the hierarchical approach to QoS support in terms of spreading network traffic avoiding focussed congestion which may be incurred in quickly overused trunks.

### *B. Path Management with Quantized Constraints*

We define a path management scheme which determines how calls are allocated to paths. This path management primarily deals with *path sets* which are collections of paths computed with the same constraints. Each path set is associated with a certain destination and a set of multiple QoS constraints, and all the elements in the path set are the results of the multiple QoS path computation for the same destination with the constraints. As the first operational aspect of running the path computation algorithms, we consider where QoS constraints are derived from and how they are bounded to corresponding path sets. QoS constraints can be either defined by applications or provided by the network. In the former, QoS constraints are assumed to be continuous variables (e.g., packet loss, delay, bandwidth). An application can choose and negotiate an arbitrary set of QoS constraints with the network. This allows applications broad flexibility in defining QoS constraints, but makes it difficult for the network to precompute and quickly provision QoS paths. For the network, it would be more convenient to enforce limited sets of QoS constraints, to which the applications must abide. We refer to this pre-definition of constraints *constraint quantization*.

The constraint quantization allows the network to limit the number of possible QoS constraint sets and forces applications to select the one that best fits their traffic characteristics. Therefore, the calls with same quantized constraints and same destination will be aggregated together (a la DiffServ) and use the same path sets. The path set for a certain quantized constraint is opened by the first call, and is shared by all subsequent calls in the same group without path recomputation. This *fast QoS provisioning* is the benefit of the constraint quantization. In the quantized option, when multiple paths are available, a further choice is available regarding *call allocation*. As a first choice, each call is allocated to a specific path in the set. When the specific call allocation to a single path is exercised, the network system performs *flow-based* call allocation. A second option is to perform *packet-based* call allocation by spreading the packets over the multiple paths regardless of the calls they belong. In our scheme, we adopted the flow-based allocation since mostly QoS support is demanded by multimedia applications and their traffic is prone to jitter which may be caused by packets sprinkled over different paths. This path management scheme is called Quantized constraint Flow-based Multiple Paths (QFMP) in [5]. This scheme is used for the call allocation and admission control in the way described in [5].

### *C. Link State Information and Path Refreshment*

HAQP guarantees QoS services *statistically* taking into account the most recent network status information. The necessary network statistics are collected and stored in the link state database at every node in the network. This link state database is to project the global link state picture of the network such that the path computation algorithm correctly produces QoS-satisfying paths. Since HAQP does not compute paths in a centralized fashion, each node in the network is responsible for producing paths originating from themselves in a distributed way. Thus, undesirable path conflicts may occur due to the lack of coordination. The performance of this low-cost approach is influenced by the freshness of the link state database, and frequent updates of the link state database are required for desirable computations, which is usually controlled by the QoS extensions of OSPF through link state flooding, but intuitively the frequent update can be deemed expensive. This expensive approach not only produces unwanted network overhead, but also consumes router processor time. Therefore, a more efficient approach substituting the naively frequent flooding mechanism is demanded. Thus, we propose a hierarchical approach to collecting link state information; global flooding, partial link state collecting, and local traffic monitoring.

#### C.1 Global Flooding

The QoS path computation primarily depends on the link state database locally available at the source. In order to make the link state database project the global picture of the entire network conditions, the global flooding mechanism is deemed to be the most effective method. We use the controlled flooding mechanism to exchange link state information as Q-OSPF does. The global flooding carries not only the conventional link state properties but also QoS metrics such as residual bandwidth, queueing delay, loss probability, etc. Since the flooding mechanism is the only way to provide the entire network information, more frequent flooding is required when better outcomes of the path computation are expected.

However, this flooding approach is expensive in terms of network resources. The global flooding overhead depends on network topology, and intuitively we can guess the fast growth of flooding overhead when the network topology becomes larger and the flooding interval is shortened. Thus, although global flooding with frequent update yields accurate QoS path computation, it eventually leads to prohibitive overhead.

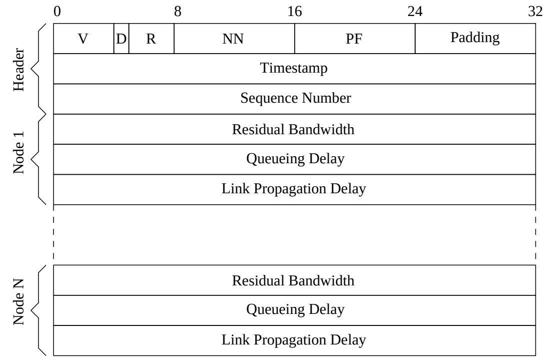


Fig. 1. HAQP-Probe packet format.

TABLE I
HAQP-PROBE PACKET FIELDS.

| | |
|---|---|
| V (4 bits) | Protocol version. |
| D (1bit) | Direction flag (0 from source to destination, 1 vice versa). |
| R (3 bits) | Reserved for future use. |
| NN (8 bits) | Number of traversed nodes. |
| PF (8 Bits) | Payload format for each routing protocol to have a different payload format and procedures reflecting proper metrics. |
| Ts (32 bits) | Time-stamp is used by the sender to explicitly calculate the round trip time or to have a better delay approximation by using metrics defined in the payload. |
| SEQ (32 bits) | Packet sequence number. |
| RB (32 bits) | Residual bandwidth. |
| QD (32 bits) | Queuing delay. |
| LpD (32 bits) | Link propagation delay. |

#### C.2 Partial Link State Collection

In order to optimize the performance/overhead behavior, a path probing mechanism (*HAQP-Probe*) has been devised. Normally, in conjunction with the multiple path computation, this scheme is to update the status of computed paths by periodically sending probing packets. This proposed probing mechanism can be implemented as a routing protocol extension (i.e., Q-OSPF), a management/control protocol extension (i.e., SNMP or ICMP), or more generally separate protocol. The HAQP-Probe mechanism consists of injecting a probing packet, with a given frequency into newly computed or already existing paths, and gathering link state information and QoS metrics (i.e., delay, residual bandwidth, etc.). Each router along the path cooperates by stamping the requested information into the probing packet. When the probing packet reaches destination, it is marked as complete and sent back to source. Both the HAQP-Probe packet format in Fig. 1 and the corresponding procedure have been designed to be compliant with different routing protocols, to be malleable for future extensions, and to provide the information and metrics requested by any particular routing protocols. This is in accordance with the principles proposed in [6]. With the common fields summarized in Table I, we designed the data format and corresponding procedures to interwork with Q-OSPF. The data format and procedures have been used in the simulations presented in section III. Besides, the residual bandwidth and queueing delay values are computed locally and are exponentially averaged in order to reduce burst impact on QoS routing information.

The HAQP-Probe mechanism suggests that intermediate routers be actively involved in propagating the link state information and QoS metrics. The detailed information propagation steps are listed in the following:

**Source :**

*Between two global flooding events*

• Probing packets for each path are sent out with a given frequency.

*When probing packets come back*

• The routing tables to the given destinations are updated accordingly.

**Intermediate nodes :**

*When each probing packet is received*

• Stamp the probing packet with the requested information according to the format defined by PF.

• Modify the NN field in the header.

• Increment the time stamp.

• Forward the packet to the next node in the path.

**Destination nodes :**

*When each probing packet is received*

• Stamp the probing packet with the requested information according to the format defined by PF.

• Modify the NN field in the header.

• Increment the time stamp.

• Reverse the direction flag in the header.

• Send the packet back to source.

The proposed mechanism can be easily implemented as an extension of management or control protocols such as SNMP or ICMP according to [7], [8], [9]. Since SNMP does not provide specific mechanism to perform an action, it is possible to use the `SetRequest` to perform a command, and an SNMP proprietary object can be used to represent a command [10]. The HAQP-Probe implementation as an SNMPv2 extension requires the definition of a new object (i.e., Probing) representing the probing action, and the probing packet is encapsulated into an SNMPv2 `SetRequest` PDU setting value of the object according to specific semantics. The implementation as an ICMP extension requires the introduction of a new ICMP packet type to handle the HAQP-Probing packet and an addendum to the protocol rules to include the HAQP-Probing rules, and this makes the approach less convenient to deploy. Using either SNMP or ICMP is not expected to burden the forwarding engine with significant additional overhead, and the HAQP-Probing scheme can be inexpensively deployed in current routers.

### C.3 Local Traffic Monitoring

At the finest level of link state monitoring, edge routers monitor their outgoing traffic through existing MPLS pipes. Since each edge router computes multiple QoS paths and keeps a database of the computed QoS paths pinned down by MPLS, by monitoring traffic through the MPLS pipes, edge routers can update the corresponding segments of their link state database. The link state of each computed path is explicitly refreshed by the probing scheme, and this local monitoring process is renewed whenever the link state of paths is updated through the probing scheme. Local monitoring allows the multiple QoS path computation algorithm to take into account the most up-to-date information of locally outgoing traffic. Although other traffic information originating from other edge routers is not provided until the probing scheme collects path-specific link state information and the global flooding entirely updates the link state database, at least the locally originating traffic can be accounted for in the path computation and CAC. Since multiple QoS paths are used in parallel, dynamic changes of network traffic are distributed over the paths and are faithfully refreshed by local monitoring, this alleviates the need of frequent probing and of global flooding.

## III. Simulation Experiments

Simulation experiments are to compare the performance of the two QoS provisioning approaches; the conventional approach and HAQP. The simulation results were generated and analyzed using SENSE which is powered by Parsec [11] and developed by the HPI group at UCLA[1]. The traffic in the simulation experiments is IP Telephony. As in [3], the peak rate of the traffic is 64 Kbps and the average rate 23 Kbps (25 Kbps including overhead).

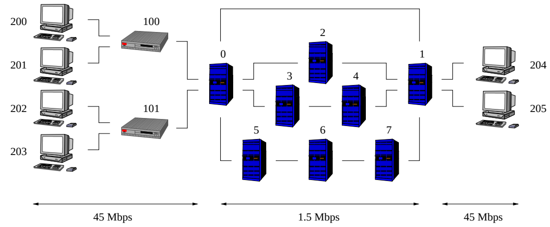


Fig. 2. Simulation topology.

TABLE II

Simulation configuration parameters.

| | |
|---|---|
| **Network properties** | |
| Link capacity | 1.5 Mbps for core links and 45 Mbps for edge links |
| Propagation delay | 1 ms for core links and 10 $\mu$sec for edge links |
| Bottleneck buffer | 18750 bytes (i.e., maximum queueing delay of 0.1 sec) |
| **Call properties** | |
| Two end nodes | nodes from 200 through 205 |
| Arrival rate | 2 / sec over nodes from 200 through 203 |
| Duration | 180 sec of exponential average |
| **Traffic QoS constraints** | |
| Residual bandwidth | 300 Kbps |
| End-to-end delay | 100 ms |
| **Simulation cases** | |
| Conventional approach | 2, 4, 8, 16, 32, 64, 128 sec flooding interval |
| HAQP | 2, 4, 8, 16, 32, 64, 128 sec probing interval and flooding interval = probing interval $\times$ 8 |

Since HAQP provides measurement-based QoS guarantee, a safety bandwidth margin must be considered on each link to protect from traffic fluctuation. In the experimental scenario, considering various simulation parameters, we chose the bandwidth margin of 300 Kbps. The violation of the bandwidth margin constraint is a symptom of performance deterioration. To compare the proposed scheme with the classical Q-OSPF approach, we compute the fraction of packets which meet the defined end-to-end delay constraints. Note that some of the total packets may violate the expected end-to-end delay and even get dropped. Here we consider a simple topology with 16 nodes as in Fig. 2. In this scenario, the routers in the core area become the bottlenecks. The QoS routing scheme considers the residual bandwidth on these links to make CAC decisions. The rest of the configuration parameters are summarized in Table II.

Each simulation case is performed with different flooding/probing intervals. The different intervals produce different overhead and different results for CAC and QoS routing. The overhead is measured as the amount of flooding/probing packets in bits over the entire simulation time. In order to see how efficiently calls are admitted, we collect the packet statistics of admitted calls. The performance of the two schemes and their overhead generated by different flooding/probing intervals are summarized in Table III. The performance degradation for fixed overhead is compared and shown in Fig. 3. When less frequent update of link state information is deployed in the conventional QoS provisioning, the performance degradation is much worse than in HAQP. Since each edge router with HAQP monitors their local outgoing traffic and locally updates their

[1] High Performance Internet research group in Network Research Laboratory (NRL) : http://www.cs.ucla.edu/NRL/hpi

TABLE III

OVERHEAD AND CAC/QOS ROUTING RESULTS FOR THE SIMULATION CASES.

| Conventional approach | | | HAQP | | |
|---|---|---|---|---|---|
| Flood intv. (sec) | Overhead (Kbps) | Satisfied packets (%) | Probe intv. (Flood intv.) (sec) | Overhead (Kbps) | Satisfied packets (%) |
| 2 | 95.4 | 100.00 | 2 (16) | 270.2 | 100.00 |
| 4 | 47.7 | 99.48 | 4 (32) | 134.9 | 100.00 |
| 8 | 23.9 | 91.34 | 8 (64) | 67.3 | 100.00 |
| 16 | 12.1 | 76.33 | 16 (128) | 33.7 | 100.00 |
| 32 | 6.0 | 58.54 | 32 (256) | 16.6 | 99.45 |
| 64 | 3.2 | 42.10 | 64 (512) | 8.5 | 99.51 |
| 128 | 1.6 | 24.56 | 128 (1024) | 3.8 | 98.24 |

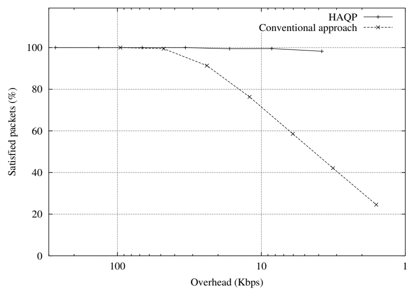


Fig. 3. Performance indices of two schemes over different network overhead.

link state database, less frequent probing and even longer flooding interval do not seriously hurt the performance. The partial link state information collection also alleviates the necessity of frequent global flooding.

We can explain the reason for the performance degradation in Fig. 3 by investigating concurrently active (i.e., admitted) calls. According to the simulation configuration, we can compute the approximate number of calls that can be accommodated by the network when the bottleneck links are saturated; the network capacity = (bottleneck link capacity - bandwidth margin) $\times$ the number of possible paths = (1.5 Mbps - 300 Kbps) $\times$ 4 = 4.8 Mbps, and this network can accommodate approximately 192 (4.8 Mbps $\div$ 25 Kbps) calls at steady state. Fig. 4 shows that when the flooding interval increases in the conventional scheme, the CAC and QoS routing make wrong decisions admitting too many calls due to out-of-date link state information, while HAQP accepts a much reasonable number of calls as shown in Fig. 5.

If the probing interval in HAQP is similar to the flooding interval in the conventional scheme, the entire overhead of HAQP may be even greater than the one of the conventional scheme as

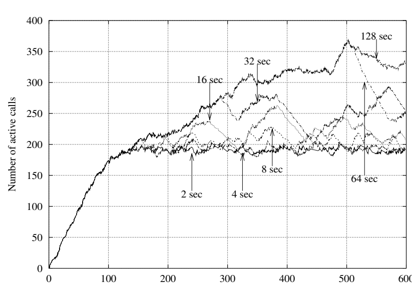


Fig. 4. Number of calls admitted by the conventional scheme with different flooding intervals.

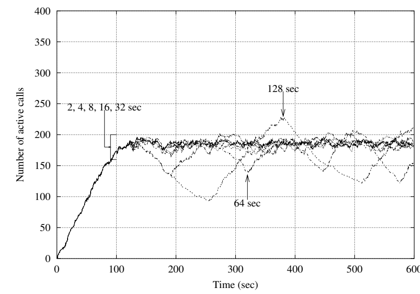


Fig. 5. Number of calls admitted by HAQP with different partial link state collecting intervals.

in Table III since HAQP deploys multiple paths and the amount of the probing packets are proportional to the number of active paths. Besides, HAQP still uses the Q-OSPF flooding mechanism to make the entire link state database up-to-date, and this results in additional overhead. However, the reason that HAQP can be characterized to be cost-effective is that for the same amount of overhead, HAQP outperforms the conventional scheme. That is, for the same performance, much less overhead is generated by required link state information acquisition.

## IV. CONCLUSION

This paper addresses a novel approach (HAQP) for QoS provisioning in IP networks. This hierarchical approach deploys multiple paths for better resource utilization, QoS constraint quantization for fast QoS provisioning and path management, and subdivided methods for cost-effective link state information acquisition. The better performance of HAQP is shown through the simulation experiments and the same performance of the conventional approach is achieved by HAQP with relatively small amount of overhead.

## REFERENCES


[1] R. Guerin, A. Orda, and D. Williams, "QoS routing mechanisms and OSPF extensions," in *Proc. of Global Internet (Globecom)*, Phoenix, Arizona, Nov. 1997.

[2] Dirceu Cavendish and Mario Gerla, "Internet QoS Routing using the Bellman-Ford Algorithm," in *IFIP Conference on High Performance Networking*, 1998.

[3] Alex Dubrovsky, Mario Gerla, Scott Seongwook Lee, and Dirceu Cavendish, "Internet QoS Routing with IP Telephony and TCP Traffic," in *Proc. of ICC*, June 2000.

[4] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," Request for Comments 3031, Internet Engineering Task Force, Jan. 2001.

[5] Scott Seongwook Lee and Mario Gerla, "Fault tolerance and load balancing in QoS provisioning with multiple MPLS paths," *Lecture Notes in Computer Science*, vol. 2092, pp. 155–, 2001, International Workshop on Quality of Service (IWQoS).

[6] David D. Clark and David L. Tennenhouse, "Architectural considerations for a new generation of protocols," Philadelphia, Pennsylvania, Sept. 1990, IEEE, pp. 200–208, Computer Communications Review, Vol. 20(4), Sept. 1990.

[7] J. C. Mogul and J. Postel, "Internet standard subnetting procedure," Request for Comments 950, Internet Engineering Task Force, Aug. 1985.

[8] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser, "Protocol operations for version 2 of the simple network management protocol (SNMPv2)," Request for Comments 1905, Internet Engineering Task Force, Jan. 1996.

[9] G. Apostolopoulos, S. Kama, D. Williams, R. Guerin, A. Orda, and T. Przygienda, "QoS routing mechanisms and OSPF extensions," Request for Comments 2676, Internet Engineering Task Force, Aug. 1999.

[10] William Stallings, "SNMP and SNMPv2: the infrastructure for network management," vol. 36, no. 3, pp. 37–43, Mar. 1998.

[11] Rajive Bagrodia, Richard Meyer, Mineo Takai, Yu an Chen, Xiang Zeng, Jay Martin, and Ha Yoon Song, "Parsec: A parallel simulation environment for complex systems," *Computer*, vol. 31, no. 10, pp. 77–85, Oct. 1998.