QoS Routing Granularity in MPLS Networks

Ying-Dar Lin and Nai-Bin Hsu, National Chiao Tung University
Ren-Hung Hwang, National Chung Cheng University

Abstract

This study investigates how constraint-based routing decision granularity significantly affects the scalability and blocking performance of QoS routing in an MPLS network. Coarse granularity, such as per-destination, has lower storage and computational overheads but is only suitable for best effort traffic. On the other hand, fine granularity, such as per-flow, provides lower blocking probability for bandwidth requests, but requires a huge number of states and high computational cost. To achieve cost-effective scalability, this study proposes using hybrid granularity schemes. The *overflowed cache* of the per-pair/flow scheme adds a per-pair cache and a per-flow cache as the routing cache, and performs well in blocking probability. *The per-pair/class* scheme groups the flows into several paths using routing marks, thus allowing packets to be label-forwarded with a bounded cache.

Introduction

The Internet provides users diverse and essential quality of service (QoS), particularly given the increasing demand for a wide spectrum of network services. Many services, previously only provided by traditional circuit-switched networks, can now be provided on the Internet. These services, depending on their inherent characteristics, require certain degrees of QoS guarantees. Many technologies are therefore being developed to enhance the QoS capability of IP networks. Among these technologies, *differentiated services* (DiffServ) [1–3] and *multiprotocol label switching* (MPLS) [4–6] are paving the way for tomorrow's QoS services portfolio.

DiffServ is based on a simple model where traffic entering a network is classified, policed, and possibly conditioned at the edges of the network, and assigned to different behavior aggregates. Each behavior aggregate is identified by a single *DS codepoint* (DSCP). At the core of the network, packets are fast forwarded according to the *per-hop behavior* (PHB) associated with the DSCP. By assigning traffic of different classes to different DSCPs, the DiffServ network provides different forwarding treatments and thus different levels of QoS.

MPLS integrates the label swapping forwarding paradigm with network layer routing. First, an explicit path, called a *label switched path* (LSP), is determined, and established using a signaling protocol. A label in the packet header, rather than the IP destination address, is then used for making forwarding decisions in the network. Routers that support MPLS are called *label switched routers* (LSRs). The labels can be assigned to represent routes of various *granularities*, ranging from as coarse as the destination network down to the level of each single flow. Moreover, numerous traffic engineering functions have been effectively achieved by MPLS. When MPLS is combined with DiffServ and constraint-based routing, they become powerful and complementary abstractions for QoS provisioning in IP backbone networks.

Constraint-based routing is used to compute routes that are subject to multiple constraints, namely explicit route and QoS constraints. Explicit routes can be selected statically or dynamically. However, network congestion and route flapping are two factors contributing to QoS degradation of flows. To reduce blocking probability and maintain stable QoS provision, dynamic routing that considers resource availability, namely QoS routing, is desired.

Once the explicit route is computed a signaling protocol, either Constraint-Based Routing Label Distribution Protocol (CR-LDP) or Resource Reservation Protocol with Traffic Engineering extension (RSVP-TE), is responsible for establishing forwarding state and reserve resources along the route. In addition, LSRs use these protocols to inform their peers of the label/FEC bindings they have made. A forwarding equivalence class (FEC) is a set of packets that will be forwarded in the same manner. Typically packets belonging to the same FEC will follow the same path in the MPLS domain.

0163-6804/02/$17.00 © 2002 IEEE

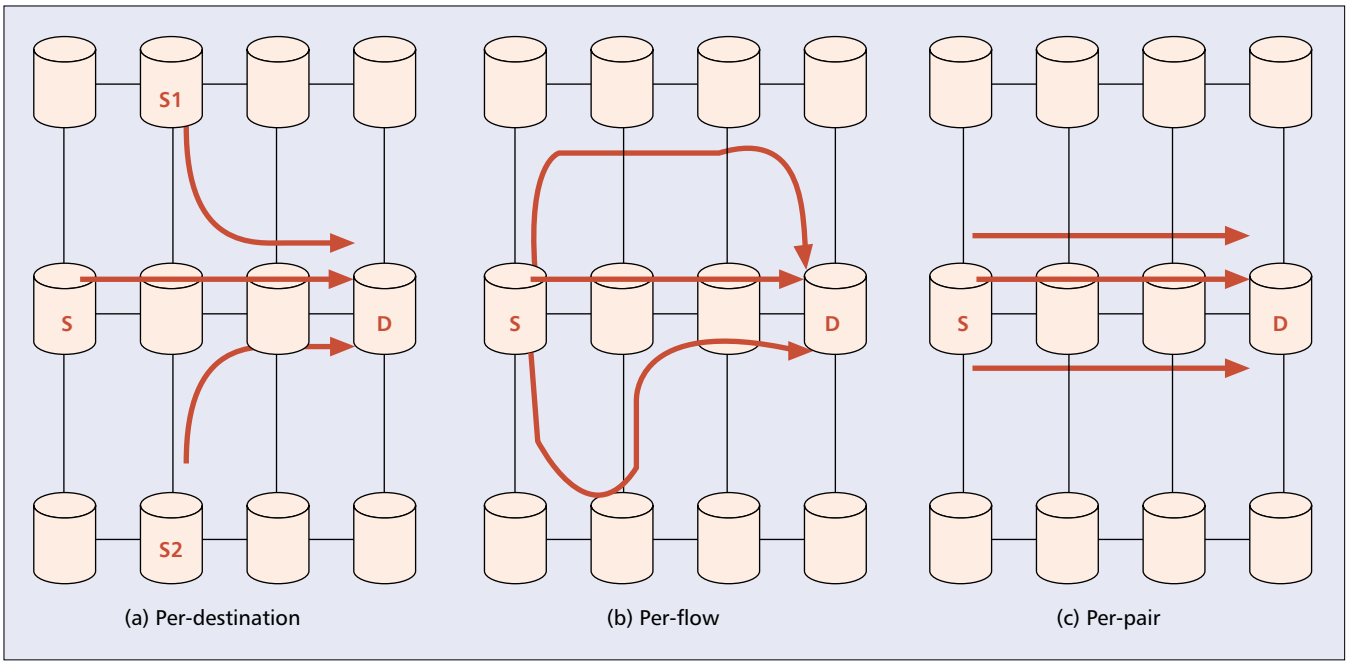

Figure 1. *An example of routing granularity.*

It is expected that both DiffServ and MPLS will be deployed in an Internet service provider's (ISP's) network. To interoperate these domains, EXP-Inferred-PSC SLP (EXP-LSP) and Label-LSP models are proposed [5]. EXP-LSP provides no more than eight bandwidth aggregate (BA) classes but scale better. On the other hand, Label-LSP provides finer service granularity but results in more state information.

The number of path computations can be reduced by path caching [7–8]. A path cache memorizes the constraint-based routing decision and behaves differently with different granularities. In coarse granularity, such as per-destination in Fig. 1a, all flows moving from different sources to a destination are routed to the same outgoing link; this has lower storage and computational overheads, but is only suitable for best effort traffic. On the other hand, with fine granularity, such as per-flow in Fig. 1b, each individual flow is computed and routed independently; this provides lower blocking probability for bandwidth requests, but requires a huge number of states and high computational cost. Figure 1c shows the per-pair granularity; all traffic between a given source and destination, regardless of the number of flows, travels the same route. Note that in cases of explicit routing, per-destination and per-pair routing decisions are identical.

This study investigates how the granularity of the routing decision affects the scalability of computation or storage, and the blocking probability of a QoS flow request. To reduce the blocking probability without sacrificing per-flow QoS requirements, two routing mechanisms are proposed from the perspective of granularity. The *Per_Pair_Flow* scheme adds a per-pair cache (P-cache) and an overflowed per-flow cache (O-cache) as a routing cache. The flows that the paths of P-cache cannot satisfy with the bandwidth requirement are routed individually and their routing decisions overflowed into the O-cache. The *Per_Pair_Class* scheme aggregates flows into a number of forwarding classes. This scheme reduces the routing cache size and is suitable for MPLS networks, where packets are labeled at edge routers and fast forwarded in the core network.

The rest of this article is organized as follows. We describe two on-demand path computation heuristics on which our study is based. We describe the overflowed cache *Per_Pair_Flow* scheme. We then describe the *Per_Pair_Class* scheme, which uses a mark scheme to reduce cache size. Subsequently, we present a simulation study that compares several performance metrics of various cache granularities. Finally, we present conclusions.

Path Computation

This article assumes that link-state-based and explicit routing architecture are used. Link state QoS routing protocols use reliable flooding to exchange link state information, enabling all routers to construct the same link state database (LSDB). Given complete topological information and the state of resource availability, each QOS-capable router finds the least costly path that still satisfies the resource requirements of a flow.

Two on-demand shortest path computation heuristics are described as the basis in this study. QoS Extensions to Open Shortest Path First (QOSPF) [9] uses the *widest shortest path* (WSP) selection criterion to select the path with the minimum number of hops (shortest). If there are several such paths, the one with the maximum available bandwidth (widest) is selected. Algorithm 1 shows the algorithm to respond to the route query. Each routing entry of $S_d^h = (D_d^h, B_d^h, \sigma_d^h)$ consists of the minimum delay D_d^h, available bandwidth B_d^h, and path σ_d^h to node d, with minimum hops h.

```
WSP_Routing (F, s, d, b, D)
topology G(V, E); /* width b_ij associate with e_ij ∈ E */
flow F;      /* from s to d with req. b and D */
routing entry S_d;
/* set of tuple(delay, width, path) from s to d */
shortest path σ;
Begin
  initialize S_d ← φ, prune e_ij if b_ij < b, ∀e_ij ∈ E
  for hop-count h = 1 to H
    B_s ← ∞, D_s ← 0
    find all paths (s, ..., x, d) with h hops, and
      Begin
      update D_d ← h
      B_d ← Max{Min[B_x, b_xd]}, ∀x
      σ_d ← σ_x ∪ d
      S_d ← S_d ∪ (D_d, B_d, σ_d)
      End
    if (S_d ≠ φ) pick path σ_d with widest B_d, stop
  endfor
End
```

Algorithm 1. *The widest shortest path (WSP) heuristic.*

```
CSP_Routing (F, s, d, b, D)
topology G(V, E) ;/* width b_ij associate with e_ij ∈ E */
flow F;          /* from s to d with req. b and D */
label L;         /* set of labeled nodes */
shortest path σ;
   /* obtained by backtracking the inspected nodes */
Begin
   1) prune e_ij if b_ij < b, ∀e_ij ∈ E
   2) initialize L ← {s}, D_i ← d_si, ∀i ≠ s
   3) find x ∉ L such that D_x = Min_{i∉L}[D_i]
      /* examine tentative nodes */
   4) if D_x > D, "path not found", stop
   5) L ← L ∪ {x}
   6) if L = V, return(σ) with delay(σ) = D_d, stop
   7) update D_i ← Min[D_i, D_x + d_xi], ∀i adjacent to x
   8) go to 3)
End
```

Algorithm 2. *The constrained shortest path (CSP) heuristic.*

This algorithm iteratively identifies the optimal (widest) paths from itself s to any other node d, in increasing order of hop count h, with a maximum of H hops, and where H can be either the value of diameter of G or can be set explicitly. Afterward, WSP picks the widest σ of all possible shortest paths to node d as the routing path with minimum hops. Thus, the complexity is $O(KH)$, where K is degree of G (i.e., the maximum degree of any node).

Another heuristic, *constrained shortest path* (CSP), shown in Algorithm 2, uses "minimum delay with abundant bandwidth" as the selection criterion to find a shortest path σ for flow F. Step 1 eliminates all links that do not satisfy the bandwidth requirement b. Next, the CSP simply finds a shortest path σ from itself (i.e., s) to destination d, as in steps 2–8. Initially, all nodes except the source are nonlabeled. When it is discovered that a node represents the shortest possible path from the source to that node, it is labeled and never visited thereafter. Step 3 chooses a nonlabeled node x with minimum delay, and x is labeled in step 5. Step 7 updates the delay metric for each adjacent node i. Meanwhile, CSP is terminated either in step 4, as the delay exceeds the threshold D before reaching destination d, or in step 6, as all nodes are labeled. Consequently, CSP finds a QoS path, $\sigma = s \ldots d$, such that $width(\sigma) \geq b$ and $delay(\sigma) \leq D$, to satisfy the bandwidth requirement b and delay requirement D. The complexity of CSP is $O(n^2)$, where $n = |V|$. Note that this complexity can be improved by using a heap data structure.

Cache with Per-Pair/Flow Granularity

This section introduces a routing scheme with per-pair/flow hybrid cache granularity. The architecture presented herein uses source routing and a hop-by-hop signaling procedure such as CR-LDP or RSVP-TE. Routes are guaranteed to be loop-free in source routing, and the signaling procedure prevents each packet of the flow from carrying complete route information. Sets of labels distinguish destination address, service class, forwarding path, and probably also privacy. In MPLS, edge devices perform most of the processor-intensive work, performing application recognition to identify flows and classify packets according to the network policies.

Upon a flow request during the signaling phase, the path query can be got through with by computing path on demand, or extracting path from the cache. When the query is successful, the source node initiates hop-by-hop signaling to set up forwarding state, and the destination node initiates bandwidth reservation backward on each link in the path.

The routing path extracted from the cache could be misleading, that is, flows following a per-destination cache entry might not find sufficient resources along the path, although there exist alternative paths with abundant resources. This lack of resources is attributed to flows of the same source-destination (S-D) pair routed on the same path led by the cache entry, which is computed merely for the first flow. Therefore, this path might not satisfy the bandwidth requirements of subsequent flows. Notably, the blocking probability increases rapidly when a link of the path becomes a bottleneck.

On the other hand, although no such misleading (assume no *staleness* of link state) occurs in per-flow routing, flow state and routing cache size could be enormous, ultimately resulting in poor scalability. Furthermore, due to the overheads of per-flow path computation, on-demand path finding is hardly feasible in real networks (with high rate requests.) Therefore, path precomputation is implemented in [9], which asynchronously computes feasible paths to destinations.

The routing cache of this scheme is functionally divided into three parts, a per-pair cache (*P-cache*), an overflowed per-flow cache (*O-cache*), and a per-destination cache (*D-cache*). To speed up access to the routing entry, caches are placed on kernel memory and never swapped out.

Shortest paths on the P-cache and D-cache are precomputed at system startup or can be flushed and computed on demand under the network administration policy. Entry of the O-cache is created when a request arrives and cannot find sufficient bandwidth on the path in P-cache. By looking up the next hop in the D-cache, best effort traffic is forwarded as in a non-QoS support OSPF router. QoS paths are extracted from the P-cache in this scheme.

Algorithm 3 shows the *Per_Pair_Flow* scheme, which is detailed as follows. When a path query with multiple constraints is executed at the ingress LSR, look up the P-cache for routing information. If the lookup is a miss, it implies that no routing path is stored for the particular request. Therefore, in this situation the *Per_Pair_Flow* invokes the *FindRouteLeastCost* function to find a QoS path σ. This path is stored in the P-cache and the flow request F is sent through σ explicitly. But if path σ cannot be found, the request is blocked.

However, if the lookup of P-cache is a hit, a resource availability check must be made according to the latest link states to ensure the QoS of the flow. If the check is successful, the signaling message of F is sent according to the P-cache. Meanwhile, if the check fails, function *FindRouteLeastCost* is invoked to find an alternative path based on the information in LSDB and the residual bandwidth database (RBDB). This QoS path σ is stored in the O-cache (i.e., *overflowed* to the O-cache), and signaling of F is sent through σ. But if path σ cannot be found, the flow is blocked.

Function *FindRouteLeastCost* in Algorithm 3 finds a QoS path on demand using WSP or CSP heuristics. The link cost function in this computation can be defined according to the needs of network administrators. For example, hop counts, exponential cost [10], or distance [11] can be used as the link cost metric in the computing function.

```
Per_Pair_Flow(F, s, d, b, D)
flow F; /* from s to d with req. b and D */
path σ,
Begin
  case miss(P-cache):
    σ ← FindRouteLeastCost(s, d, b, D)
    if (σ found)
      insert (P-cache), label(F) & route(F) through σ
    else "path not found"
  case σ ← hit(P-cache):
    if (width(σ) ≥ b) and (delay(σ) ≤ D)
      label(F) & route(F) through σ
    else /* overflow */
      Begin
      σ ← FindRouteLeastCost(s, d, b, D)
      if (σ found)
        insert(O-cache), label(F) & route(F) through σ
      else "path not found"
      End
End
```

Algorithm 3. Per_Pair_Flow *routing.*

Src., dst.	LSP_1	…	LSP_m
s,d_1	π_{11}, $width_{11}$, $delay_{11}$, ρ_{11}	…	π_{1m}, …
s,d_i	π_{l1}, $width_{l1}$, ρ_{l1}	…	…

Table 1. *Routing cache in* Per_Pair_Class *routing.*

Cache with Per-pair/class Granularity

This section presents another hybrid granularity scheme using a routing *mark* as part of the label in MPLS. Herein, when a flow request arrives at an edge router it is routed to the nearly best path given the current network state, where the "best" path is defined as the least costly feasible path. Flows between an S-D pair are routed on several different paths and marked accordingly at the source and edge routers. Notably, flows of the same routing path may require different QoS. The core router in an MPLS domain uses the label to determine to which output port (interface) a packet should be forwarded, and to determine service class. Core devices expedite forwarding while enforcing QoS levels assigned at the edge.

By limiting the number of routing marks, say to m, the routing algorithm can route flows between each S-D pair along a limited number of paths. The route *pinning* is enforced by stamping packets of the same flow with the same *mark*. Rather than identifying every single flow, the forwarding process at intermediate or core routers is simplified by merely checking the label. The size of the routing cache is bounded to $O(n^2m)$, where n is the number of network nodes. Note that if the constraint-based routing is distributed at the edge nodes, each node caches m paths to other $n - 1$ destinations, and this bound reduces to $O(nm)$.

Table 1 illustrates the structure of the routing cache, which provides a maximum of m feasible routes per node pair. The first path entry LSP_1 can be precomputed, or the path information can be flushed and computed on demand under the network administration policy. Besides the path list of LSP, each path entry includes the residual bandwidth (*width*), maximum delay (*length*), and utilization (ρ). Information on the entry can be flushed by the management policy (e.g., refresh timeout or reference counts). Regarding labeling and forwarding, the approach is scalable and suitable for the DiffServ and MPLS networks.

Algorithm 4 shows that, upon a flow request F, the *Per_Pair_Class* algorithm first attempts to extract the least costly feasible path π from the routing cache. If the extraction is negative, the scheme attempts to compute the least costly feasible path, termed σ. If σ is found, *Per_Pair_Class* assigns a new mark to σ, inserts this new mark into the routing cache, and then labels/routes the flow request F explicitly through σ. Meanwhile, if π is found and the path is only lightly utilized, the *Per_Pair_Class* marks the flow F and routes it to path π. Otherwise, the flow is blocked. If the utilization of path ρ(π) exceeds a predefined threshold, the *Per_Pair_Class* can either route F to a π held in the cache,

```
Per_Pair_Class(F, s, d, b, D)
flow F;     /* from s to d with req. b and D */
cache entry Π(s, d);/* set of routing paths from s to d
*/
extracted path π;
computed path σ;
Begin
   initiate cost(NULL) ← ∞
   extract π ∈ Π (s, d)
     that cost(π) is the least & satisfy constraint
       { width(π) ≥ b, length(π) ≤ D, ... }
   case (π not found):
     σ ← FindRouteLeastCost(s, d, b, D)
     if (σ not found) then "path not found"
     insert/replace(σ, Π (s, d)),
     label(F) & route(F) through σ
   case (π is found):
     if (π(σ) lightly utilized) then
       label(F) & route(F) to π
     endif
     σ ← FindRouteLeastCost(s, d, b, D)
     if (σ not found) then "path not found"
     if (cost(σ) < cost(π)) then /* σ better */
       insert/replace(σ, Π(s, d)),
       label(F) & route(F) through σ
     else /* π better */
       label(F) & route(F) to π
     endif
```

Algorithm 4. Per_Pair_Class *routing with marks.*

or route F through a newly computed path σ, whichever is least costly. Therefore, traffic flows can be aggregated into the same forwarding class (FEC) and labeled accordingly at the edge routers. Notably, flows of the same FEC may require different service class. Consequently, flows between an S-D pair may be routed on a maximum of m different paths, where m is the maximum number of routing classes. In the *Per_Pair_Class* algorithm, function *FindRouteLeastCost* compute the least cost path using the CSP or WSP heuristics.

Performance Evaluation

This section evaluates the performance of unicast QoS routing, and particularly its sensitivity to various routing cache granularities. The performance of the proposed *Per_Pair_Flow* and *Per_Pair_Class* schemes is evaluated.

Network and Traffic Model

Simulations were run on 100-node random graphs based on Waxman's model [12]. In this model, n nodes are randomly distributed over a rectangular coordinate grid, and the distance between each pair of nodes is calculated with the *Euclidean* metric. Then edges are introduced between pairs of nodes, u, υ, with a probability depending on the distance between u and υ. The average degree of nodes in these graphs is in the range [3.5, 5]. Each link is assumed to be STM-1 or OC-3 with 155 Mb/s.

The simulations herein assume that the token rate of a leaky bucket mechanism is used as the bandwidth requirement, which is the primary metric. Furthermore, this study assumes that there are two types of QoS traffic: GS_1 has a mean rate of 3 Mb/s, while GS_2 has a mean rate of 1.5 Mb/s. The flow arrival process is assumed to be independent at each node, following a Poisson model. Flows are randomly destined to the else nodes. The holding time of a flow is assumed to be exponentially distributed with mean μ. The mean holding time can be adapted to keep the offered load at a constant. The link states of adjacent links of a source router are updated immediately, while the states of other links are updated by periodically receiving link state advertisements (LSAs). Note that the variability of flow durations, arrival processes, and bandwidth requirements complicates the simulation work by making convergence unlikely within acceptable simulation time. Alternative traffic models such as PARETO or heavy-tailed can be found in [10].

Performance Metrics

From the perspective of cache granularity, this study expects to find QoS routing techniques with a small blocking probability while maintaining scalable computational costs and storage overheads. Thus, several performance metrics are interesting here:

- Request bandwidth blocking probability, P_{req}, is defined as

$$P_{req} = \frac{\sum rejected\_bandwidth}{\sum requested\_bandwidth} = P_{rout} + P_{sig}. \quad (1)$$

P_{rout} is the routing blocking probability, defined as the probability that a request is blocked due to no existing path with sufficient resources, regardless of cache hit or miss. P_{sig} denotes the signaling blocking possibility, the probability that a successfully routed flow gets rejected during the actual backward reservation process, during the receiver-initiated reservation process of RSVP.

- Normalized routing cache size, $\widetilde{N}_{cache}$, is the storage overhead per flow for a caching scheme.
- Normalized number of path computations, $\widetilde{N}_{comp}$, is the number of path computations per flow in the simulated network.

Simulation Results

The simulation results are mainly to examine the behavior of the flows under moderate traffic loading (e.g., $\rho = 0.7$) where most of the blocking probabilities would not go beyond 20 percent. Moreover, the 95 percent *confidence interval* lies within 5 percent of the simulation average for all the statistics reported here.

Blocking Probability — This experiment focuses on the effects of inaccurate link-state information, due to their update periods, on the performance and overheads of QoS routing. Theory and algorithms of QoS routing with inaccurate information can be found in [13, 14]. Figure 2a shows the blocking probabilities, P_{req}, on the 100-node random graph with an offered load $\rho = 0.7$; the flow arrival rate $\lambda = 1$; the mean holding time is adjusted to fix the offered load; the refresh timeout of cache entry (*flush*)

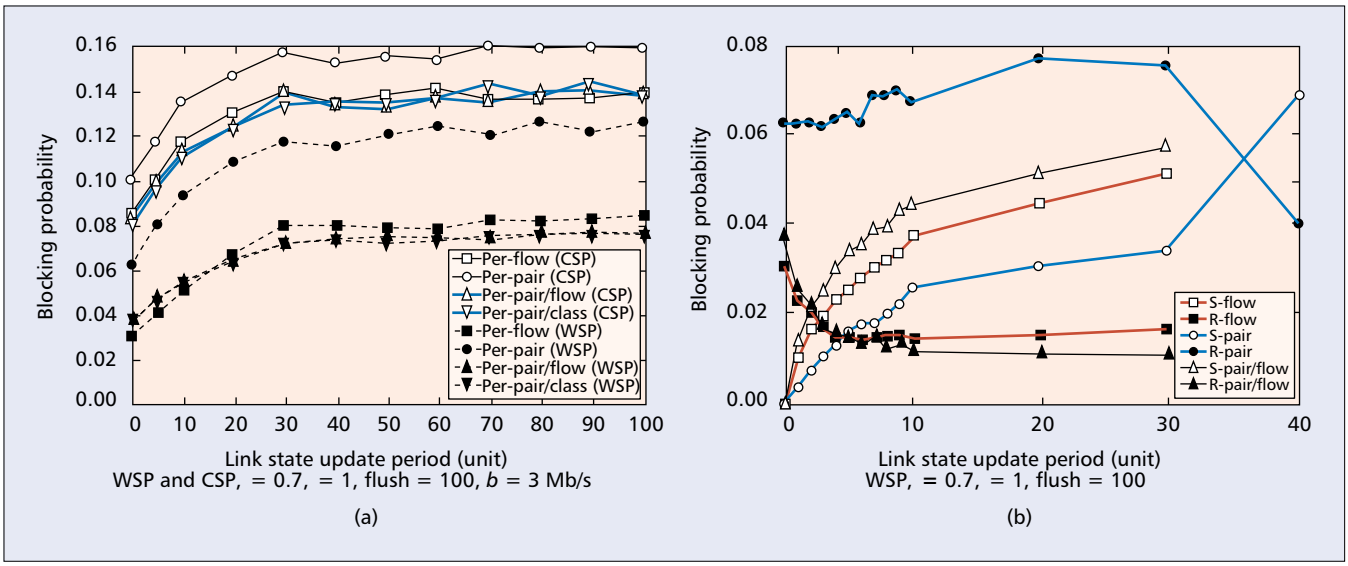

Figure 2. *Blocking probability: a) WSP vs. CSP; b) routing vs. signaling.*

= 100 units. As described earlier, CSP and WSP heuristics are used. As expected, larger update periods basically increase flow blocking. The larger update period results in a higher degree of inaccuracy in the link state, and more changes in the network could be unnoticed. As links approach saturation under the inaccuracy, LSDBs and RBDBs viewed from the source router are likely unchanged, and might mistake an infeasible path as feasible. In that case, flows are blocked in the signaling phase and only admitted if other flows leave.

However, blocking does not appear to grow even higher as the update period goes beyond a critical value. Since the larger link state update does not respond quickly enough to variations in network state, some changes are unnoticed. After many scenarios are simulated, we found that climbing of curves in longer mean holding time of flows grows slower than in shorter mean holding time of flows. This phenomenon suggests that to get more accurate network state and better QoS routing performance, the update period (the value *MaxLSInterval* in OSPF) should not go beyond the mean holding time of the admitted flows.

Per-pair routing gets higher blocking probability than other granularities. Traffic in the pure per-pair network tends to form bottleneck links and is more imbalanced than in other networks. Conversely, in the per-flow and per-pair/flow networks, the traffic obtains a QoS path more flexibly and has more chances to get alternative paths in large networks.

Intuitively, the finest granularity per-flow scheme should result in the lowest blocking probability. However, it is not always true in our experiments. In Fig. 2a, indeed, the per-flow scheme with CSP has the strongest path computation ability; it could find a feasible route for a flow under heavy load but with a longer length. A flow with a longer path utilizes more network resources than a flow with a shorter path. Although we limit the number of hops, H, of the selected path to the network diameter, the per-flow scheme still admits as many flows as it can. Eventually, network resources are exhausted; a new incoming flow is only admitted if other flows are terminated. This is why the per-flow scheme performs similar to or somewhat poorer than the per-pair/flow and per-pair/class schemes we proposed.

In addition, with the large update periods, stale link state information reduces the effectiveness of path computation of the per-flow scheme. It is possible to mistake an infeasible path as feasible (*optimistic leading*), or mistake a feasible path as infeasible (*pessimistic leading*). Thus, it will get more *signaling blocks* in the former case and *routing blocks* in the latter; both are negative to the performance of per-flow routing.

Obviously, by comparing the statistics of CSP with WSP, WSP performs better than CSP in this experiment. WSP uses breadth-first search to find multiple shortest paths and pick one with the widest bandwidth, and achieves some degree of load balancing. On the other hand, traffic is more concentrated in CSP computation networks. To cope with this shortage in CSP, appropriate link cost functions that consider the available bandwidth of the link should be chosen. Studies regarding this issue can be found in [15].

In Fig. 2b, we have insight into the blocking probability, blocked in either the routing phase (prefix R–) or the signaling phase (prefix S–). Look at the performance under accurate network state (i.e., period = 0); the *routing blocks* account for all blocked flow requests. As the update period gets larger, more and more flows mistake an infeasible path as feasible. Therefore, those flows cannot reserve enough bandwidth in the signaling phase and will be blocked. Blocking shifting from the routing to the signaling phase is caused by the staleness of network state. Rising (P_{sig}) and falling (P_{rout}) curves of each scheme cross over. The cross point is postponed in the per-pair cache scheme. A caching mechanism usually does not reflect accurate network state immediately; thus, sensitivity to staleness is reduced.

This experiment also studies the effectiveness

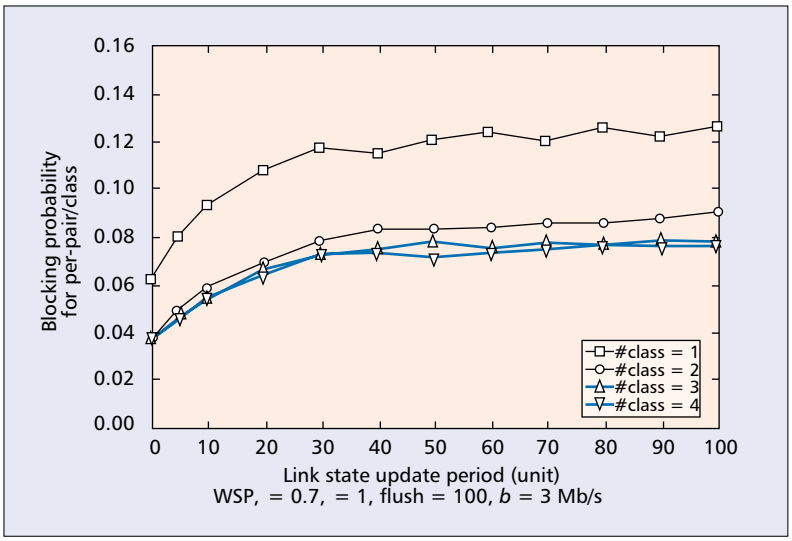

Figure 3. *The blocking probability of per-pair/class.*

of different numbers of routing classes, m, of *Per_Pair_Class* with per-pair/class granularity. Figure 3 illustrates that when $m = 1$, all flows between the same S-D pair share the same path, just the same as per-pair. When $m = 2$, the per-pair/class shows its most significant improvement over $m = 1$, but there is very little improvement when $m \geq 3$. The simulation results reveal that the *Per_Pair_Class* can yield good performance with only a very small number of routing alternatives.

Cache Size — Figure 4a gives the average number of cache entries for each single flow (i.e., normalized $\widetilde{N}_{cache}$). It indicates that the $\widetilde{N}_{cache}$ of per-flow, per-pair, and per-pair/class schemes remain nearly constant regardless of traffic loading. On the other hand, $\widetilde{N}_{cache}$ of per-pair/flow increases as traffic load increases. Statistics in Fig. 4a can be verified by the storage complexities as follows.

The cache size of per-pair is bounded by $(n - 1)^2$ with $O(n^2)$ complexity, where n is the number of nodes. Metric $\widetilde{N}_{cache}$ (per-pair) is relative to the network size and forwarding capacity. Assume the wire-speed router has forwarding capacity of 100,000 flows; $\widetilde{N}_{cache}$ (per-pair) is near to 0.1. Similarly, the cache per-pair/class is bounded by $(n - 1)^2 m$ and has a complexity of $O(n^2 m)$, where m is the number of classes. $\widetilde{N}_{cache}$ (per-pair/class) is $(n - 1)^2 m$ divided by the number of forwarding flows.

Cache granularity	Comp. overheard	Storage overhead	Blocking
Per-pair	★★★	★★★	★
Per-flow	★′	★	★★★
Per-pair/flow	★★	★★	★★★
Per-pair/class	★★★	★★★	★★★
★★★: good, ★★: medium, ★: poor, ★′ can be improved by using path precomputation.			

Table 2. *A summary of the simulation results.*

Finally, $\widetilde{N}_{cache}$ (per-flow) = 1 in Fig. 4a; thus, the cache size increases dramatically as the number of flows increases in the per-flow scheme, which disallows it to scale well for large backbone networks. Compared to per-flow, hybrid granularity is used in the per-pair/flow and the per-pair/class schemes, both significantly reducing the cache size to about 10 percent (in light load) to 20–40 percent (in heavy load) without increasing the blocking probability, compared to the per-flow in Fig. 2a.

Number of Path Computations — Figure 4b compares the average number of path computations per flow (i.e., normalized $\widetilde{N}_{comp}$) of various schemes. This metric primarily evaluates the computational cost. Note that in order to evaluate the effect of granularity in QoS routing, the simulation only uses on-demand WSP and CSP path computation heuristics. However, only plotting curves of WSP are shown; statistics of CSP

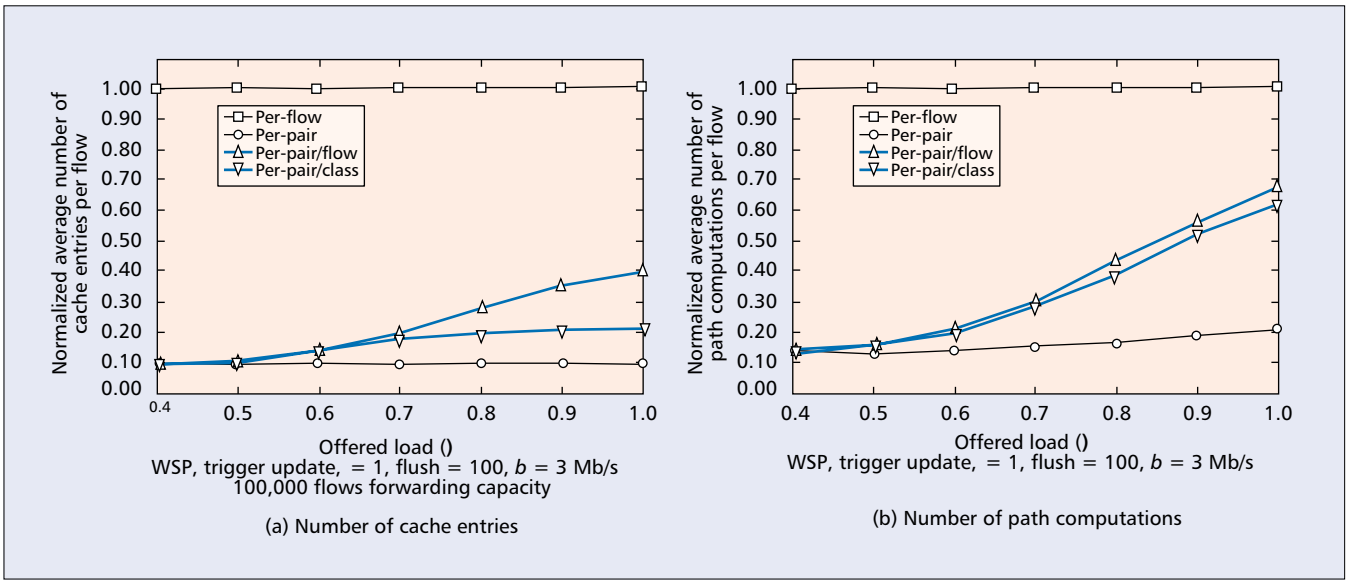

Figure 4. *Average cost per flow: a) number of cache entries; b) number of path computations.*

are almost the same as WSP. Obviously, Fig. 4b has an upper bound (i.e., $\widetilde{N}_{comp}$ (per-flow) = 1) and a lower bound (i.e., $\widetilde{N}_{comp}$ (per-pair)) which increases as the number of blocked flows increases. Note that the $\widetilde{N}_{comp}$ (per-pair/flow) and $\widetilde{N}_{comp}$ (per-pair/class) are quite influenced by the refresh timeout of entry (i.e., *flush*).

CONCLUSIONS

This study investigates how granularity affects constraint-based routing in MPLS networks and proposes hybrid granularity schemes to achieve cost effective scalability. The *Per_Pair_Flow* scheme with *per-pair/flow* granularity adds a *P-cache* (per-pair) and an *O-cache* (per-flow) as the routing cache, and performs low blocking probability. The *Per_Pair_Class* scheme with *per-pair/class* granularity groups the flows into several routing paths, thus allowing packets to be label-forwarded with a bounded cache size.

Extensive simulations are run with various routing granularities, and the results are summarized in Table 2. Per-pair cache routing has the worst blocking probability because the coarser granularity limits the accuracy of the network state. *Per-pair/flow* granularity strengthens the path-finding ability just as per-flow granularity does. Additionally, *per-pair/class* granularity has small blocking probability with a bounded routing cache. Therefore, this scheme is suitable for constraint-based routing in MPLS networks.

ACKNOWLEDGMENT

The authors wish to acknowledge the constructive comments from the anonymous reviewers.

REFERENCES

[1] D. Black *et al.*, "An Architecture for Differentiated Services," RFC 2475, Dec. 1998.
[2] K. Nichols *et al.*, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," RFC 2474, Dec. 1998.
[3] T. Li and Y. Rekhter, "A Provider Architecture for Differentiated Services and Traffic Engineering (PASTE)," RFC 2430, Oct. 1998.
[4] E. Rosen and A. Viswanathan and R. Callon, "Multiprotocol Label Switching Architecture," RFC 3031, Jan. 2001.
[5] F. Le Faucheur, "MPLS Support of Differentiated Services," draft-ietf-mpls-diff-ext-02.txt, Oct. 1999.
[6] E. Rosen *et al.*, "MPLS Label Stack Encoding," RFC 3032, Jan. 2001.
[7] M. Peyravian and A. D. Kshemkalyani, "Issues, Algorithms and a Simulation Study," *Perf. Eval.*, vol. 20. no. 8, 1997, pp. 605–14.
[8] G. Apostolopoulos *et al.*, On Reducing the Processing Cost of On-Demand QoS Path Computation," *Proc. Int'l. Conf. Net. Protocols*, Austin, TX, Oct. 1998.
[9] G. Apostolopoulos *et al.*, "QoS Routing Mechanisms and OSPF Extensions," RFC 2676, Aug. 1999.
[10] J. A. Shaikh, "Efficient Dynamic Routing in Wide-Area Networks," Ph.D. thesis, Univ. of MI, May 1999.
[11] Q. Ma, P. Steenkiste, and H. Zhang, "Routing High-Bandwidth Traffic in MaxMin Fair Share Networks," *SIGCOMM '96*, Aug. 1996.
[12] B. M. Waxman, "Routing of Multipoint Connections," *IEEE JSAC*, vol. 6, no. 9, Dec. 1988, pp. 1617–22.
[13] R. Guerin and A. Orda, "QoS-Based Routing in Networks with Inaccurate Information: Theory and Algorithms," *IEEE/ACM Trans. Net.*, vol 7, no. 3, Aug. 1999, pp. 605–14.
[14] E. Felstaine, R. Cohen, and O. Hadar, "Crankback Prediction in Hierarchical ATM Networks," *Proc. INFOCOM '99*, IEEE, July 1999.
[15] D. Zappala, D. Estrin, and S. Shenker, "Alternate Path Routing and Pinning for Interdomain Multicast Routing," Tech. rep. 97–655, USC, 1997.

BIOGRAPHIES

YING-DAR LIN (ydlin@cis.nctu.edu.tw) received his B.S. degree in computer science and information engineering from National Taiwan University in 1988, and his M.S. and Ph.D. degrees in computer science from the University of California, Los Angeles in 1990 and 1993, respectively. He joined the faculty of the Department of Computer and Information Science at National Chiao Tung University in August 1993. His research interests include design, analysis, and implementation of network protocols and algorithms, wire-speed switching and routing, and intranet servers.

NAI-BIN HSU (naibin@ms7.hinet.net) received his B.S. degree from National Taiwan Institute of Technology and his M.S. degree in computer science and information engineering from National Chiao Tung University, Taiwan, in 1984 and 1990, respectively. He joined Telecom Lab in 1984, where he worked on the National Information Infrastructure project. His research interests include of QoS routing and performance evaluation.

REN-HUNG HWANG (rhhwang@cs.ccu.edu.tw) received his B.S. degree in computer science and information engineering from National Taiwan University in 1985, and his M.S. and Ph.D. degrees in computer science from the University of Massachusetts at Amherst in 1989 and 1993, respectively. He joined the faculty of the Department of Computer Science and Information Engineering, at National Chung Cheng University, Chiayi, Taiwan in August 1993. His research interests include design, analysis, and implementation of Internet technology and multimedia on demand.

Per-pair/class granularity has small blocking probability with a bounded routing cache. Therefore, this scheme is suitable for constraint-based routing in MPLS networks.