Granularity of QoS Routing in MPLS Networks

Ying-Dar Lin[1], Nai-Bin Hsu[1], and Ren-Hung Hwang[2]

[1] Dept of Computer and Info. Science, National Chiao Tung University, Hsinchu, Taiwan.
{ydlin, gis84811}@cis.nctu.edu.tw
[2] Dept of Computer Science and Info. Eng., National Chung Cheng University, Chiayi, Taiwan.
rhhwang@cs.ccu.edu.tw

Abstract. This study investigates how the Constraint-based routing decision granularity significantly affects the scalability and blocking performance of QoS routing in MPLS network. The coarse-grained granularity, such as per-destination, has lower storage and computational overheads but is only suitable for best-effort traffic. On the other hand, the fine-grained 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 with a reasonable overflow ratio of 10% as offered load=0.7. *Per-pair/class* scheme groups the flows into several paths using routing marks, thus allowing packets to be label-forwarded with a bounded cache.

1 Introduction

The Internet is providing users diverse and essential Quality of Services (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, the *Differentiated Services* (DiffServ) [1–4] and *Multi-Protocol Label Switching* (MPLS) [5–8] are the enabling technologies that are paving the way for tomorrow's QoS services portfolio.

The 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 the *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 constraints 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 Label Distribution Protocol (CR-LDP) or RSVP extension (RSVP-TE), is responsible for establishing forwarding state and reserve resources along the route. In addition, LSR use these protocols to inform their peers of the label/FEC bindings they have made. Forwarding Equivalence Class (FEC) is a set of packets which will be forwarded in the same manner. Typically packets belonging to the same FEC will follow the same path in the MPLS domain.

It is expected that both DiffServ and MPLS will be deployed in ISP's network. To interoperate these domains, EXP-LSP and Label-LSP models are proposed [7]. EXP-LSP provides no more than eight Behavior Aggregates (BA) classes but scale better. On the other hand, Label-LSP provides finer service granularity but results in more state information.

Path cache [9] memorizes the Constraint-based routing decision and behaves differently with different granularities. The coarse-grained granularity, such as per-destination, all flows moving from different sources to a destination are routed to the same outgoing link, has lower storage and computational overheads but is only suitable for best-effort traffic. On the other hand, the fine-grained granularity, such as per-flow, each individual flow is computed and routed independently, provides lower blocking probability for bandwidth requests, but requires a huge number of states and high computational cost. In 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 requirement, two routing mechanisms are proposed from the perspective of granularity. The *Per_Pair_Flow* scheme adds a per-pair cache (P-cache) and an over-

flowed per-flow cache (O-cache) as 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 paper is organized as follows. Section 2 describes two on-demand path computation heuristics which our study based on. Section 3 describes the overflowed cache *Per_Pair_Flow* scheme. Section 4 then describes the*Per_Pair_Class* scheme, which uses a mark scheme to reduce cache size. Subsequently, Sect. 5 presents a simulation study that compares several performance metrics of various cache granularities, Finally, Sect. 6 presents conclusions.

2 Path Computation

WSP_Routing(F, s, d, b, D)
topology $G(V, E)$; /* width b_{ij} associate with $e_{ij} \in E$ */
flow F; /* from s to d with req. b and D */
routing entry S_d;
/* set of tuple$(length, width, path)$ from s to d */
shortest path σ;
Begin
  initialize $S_d \leftarrow \phi$, prune e_{ij} **if** $b_{ij} < b, \forall e_{ij} \in E$
  for hop-count $h = 1$ to H
    $B_s \leftarrow \infty$, $D_s \leftarrow 0$
    find all paths $(s, ..., x, d)$ with h hops, and
      Begin
      update $D_d \leftarrow h$
      $B_d \leftarrow Max\{Min[B_x, b_{xd}]\}, \forall x$
      $\sigma_d \leftarrow \sigma_x \cup d$
      $S_d \leftarrow S_d \cup (D_d, B_d, \sigma_d)$
      End
    if $(S_d \neq \phi)$ pick path σ_d with widest B_d, **stop**
  endfor
End

Fig. 1. Widest-Shortest Path (WSP) heuristic

This paper 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. QOSPF [10] uses the *Widest-Shortest Path* (WSP) selection criterion. Figure 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 shortest length D_d^h, width B_d^h and path σ_d^h to node d, with minimum hops h.

This algorithm iteratively identifies the optimal (widest) paths from itself (i.e. s) to any 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. Afterwards, WSP picks the widest σ of all possible shortest paths to node d as the routing path with minimum hops.

CSP_Routing(F, s, d, b, D)
topology $G(V, E)$;/* width b_{ij} associate with $e_{ij} \in E$ */
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** $b_{ij} < b, \forall e_{ij} \in E$
2) initialize $L \leftarrow \{s\}, D_i \leftarrow d_{si}, \forall i \neq s$
3) find $x \notin L$ such that $D_x = Min_{i \notin L}[D_i]$
/* examine tentative nodes */
4) **if** $D_x > D$, "path not found", **stop**
5) $L \leftarrow L \cup \{x\}$
6) **if** $L = V$, return(σ) with $length(\sigma) = D_d$, **stop**
7) update $D_i \leftarrow Min[D_i, D_x + d_{xi}], \forall i$ adjacent to x
8) go to 3)
End

Fig. 2. Constrained Shortest Path (CSP) heuristic

Another heuristic, *Constrained Shortest Path* (CSP), shown in Fig. 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. Step 3 chooses a non-labeled node x with minimum length and x is labeled in step 5. Step 7 updates the length metric for each adjacent node i. Meanwhile, CSP is terminated either in step 4, as the length 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 $length(\sigma) \leq D$, to satisfy the bandwidth requirement b and length requirement D.

3 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 hop-by-hop signaling procedure such as CR-LDP or RSVP-TE. Loop-free can be guaranteed

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 through with by computing path on-demand, or extracting path from the cache. When the query is successful, the source node initiates the hop-by-hop signaling to setup forwarding state and destination node initiates bandwidth reservation backward on each link in the path.

The routing path extracted from the cache could be *misleading*, i.e., 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 are 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) occur in the 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 pre-computation is implemented in [10], which asynchronously compute 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*). Shortest paths on the P-cache and the D-cache are pre-computed at the system start-up 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.

Figure 3 shows the *Per_Pair_Flow* scheme, is detailed as follows. When a path query with multiple constraints is executed at the ingress LSR, lookup 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. If the path σ is found, this path is stored in the P-cache and the flow request F is sent through σ explicitly. Otherwise, if no path can be found, the request is blocked.

However, if the lookup of the 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 on the Residual Bandwidth Database (RBDB). If a QoS path σ is found, the path is stored in the O-cache, i.e. *overflowed* to the O-cache and signaling of F is sent through σ. Finally, if no path can be found, the flow is blocked.

```
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
```

Fig. 3. *Per_Pair_Flow* routing

Function *FindRouteLeastCost* in Fig. 3 on-demand finds a QoS path using WSP or CSP heuristics in Sect. 2. The link cost function in this computation can be defined according to the needs of network administrators. For example, hop counts, exponential cost [11], or distance [12] can be used as the link cost metric in the computing function.

Assuming a flow arrival F from s to d with requirement b, the probability of overflowing the P-cache, namely θ, can be defined as $\theta = Prob(width(\sigma) < b)$, where σ is the path between s and d held in the P-cache. Simulation results show that θ is between zero to 0.3 in a 100 nodes network, depending on the offered load in the *Per_Pair_Flow* scheme. For example, θ is 10% as offered load ρ=0.7, if the forwarding capacity of the router is $100K$ flows, the *Per_Pair_Flow* scheme can reduce the number of routing cache entries from $100K$ to $20K$, including P-cache and O-cache. Additionally, number of cache entries could be further bounded by the scheme, *Per_Pair_Class*, presented in Sect. 4.

4 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 qualities of service. The core router in a MPLS domain uses the label to determine which output port (interface) a packet should be forwarded to, 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, this bound reduce to $O(nm)$.

Figure 4 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 pre-computed, 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, for instance, refresh timeout or reference counts. Regarding labeling and forwarding, the approach is scalable and suitable for the *Differentiated Services* and MPLS networks.

src, dst	LSP_1	.....	LSP_m
s, d_1	π_{11}, $width_{11}$, $delay_{11}$, ρ_{11}	.....	π_{1m}, ...
s, d_i	π_{i1}, $width_{i1}$, $delay_{i1}$, ρ_{i1}	.....	.....

Fig. 4. Routing cache in the *Per_Pair_Class* routing

Figure 5 describes 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

```
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 (ρ(π) 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
End
```

Fig. 5. *Per_Pair_Class* routing with marks

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 $\rho(\pi)$ exceeds a pre-defined threshold, the *Per_Pair_Class* can either route F to a π held in the cache, 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 *constrained shortest path* or *widest-shortest path* heuristics in Sect. 2.

5 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 are evaluated.

5.1 Network and Traffic Model

Simulations were run on 100-node random graphs based on the Waxman's model [13]. 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, v, with a probability depending on the distance between u and v. 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 155Mbps.

The simulations herein assume that the token rate 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 Mbps, while GS_2 has a mean rate of 1.5 Mbps. 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).

5.2 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: (1) Request bandwidth blocking probability, P_{req}, is defined as

$$P_{req} = \frac{\sum rejected\_bandwidth}{\sum rquest\_bandwidth} = P_{rout} + P_{sig} \quad . \tag{1}$$

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, namely the probability that a successfully routed flow gets rejected during the actual backward reservation process, during the *receiver-initiated* reservation process of RSVP. (2) Cache misleading probability, P_{misl}, is the probability of a query hit on the routing cache but the reservation signaling being rejected due to insufficient bandwidth. (3) Normalized routing cache size, or $\widetilde{N}_{cache}$, is the storage overhead per flow for a caching scheme. (4) Normalized number of path computations, or $\widetilde{N}_{comp}$, is the number of path computations per flow in the simulated network.

5.3 Simulation Results

The simulation results are mainly to examine the behavior of the flows under moderate traffic loading (e.g., ρ=0.7) where most of the blocking probabilities would not go beyond 20%. Moreover, the 95% *confidence interval* lies within 5% 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. Figure 6 and 7 show the blocking probabilities, P_{req}, on the 100-node random graph with an offered load ρ=0.7; the flow arrival rate λ=1; the mean holding time is adjusted to fix the offered load; the refresh timeout of cache entry (*flush*)=100 units. As described in Sect. 2, CSP and WSP heuristics are used. As expected, larger update periods basically increase flow blocking. The larger update period results in the higher degree of inaccuracy in the link-state, and more changes in network could be unnoticed. As links approach saturated under the inaccuracy, link states and residual bandwidth databases viewed from the source router are likely unchanged, might mistake infeasible path as feasible. In that case, flows are blocked in signaling phase and only admitted if other flows leave.

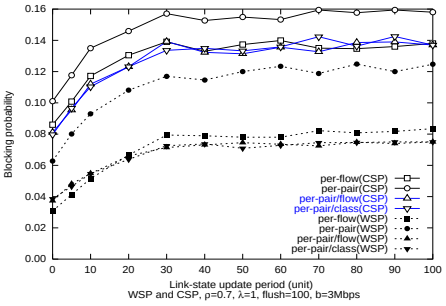

Fig. 6. Blocking probability with large requirement

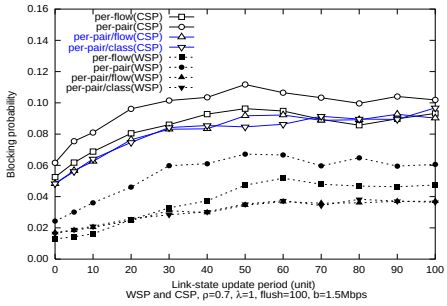

Fig. 7. Blocking probability with small requirement

However, blocking does not appear to grow even higher as the update period goes beyond a critical value. Figure 7 shows results of the experiment as in Fig. 6 but with the less bandwidth requirement and longer mean duration of the flow. The climbing of the curves grow slower than those in Fig. 6. This phenomenon suggests that to get more accurate network state and better QoS routing performance, update period (namely 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 tend 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. 6, 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 longer path utilizes more network resources than a flow with

shorter path. Though we limit the number of hops, namely 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, new incoming flow is only admitted if other flows are terminated. That is why the per-flow scheme performs similarly to or somewhat poor than the per-pair/flow and per-pair/class schemes that 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 possibly to mistake infeasible path as feasible (*optimistic-leading*), or mistake feasible path as infeasible (*pessimistic-leading*). Thus, it will get more *signaling blocks* in former case and *routing blocks* in latter case; both are negative to the performance of per-flow routing.

Obviously, by comparing the statistics of CSP with WSP, WSP heuristic 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 which consider the available bandwidth of the link should be chosen. Studies with regarding to this issue can be found in [14].

This experiment also studies the effectiveness of different numbers of routing classes, m, of *Per_Pair_Class* with per-pair/class granularity. Figure 8 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 compared to $m = 1$, but there is very little improvement when $m \geq 3$. The simulation results reveal that the *Per_Pair_Class* can yield a good performance with only a very small number of routing alternatives.

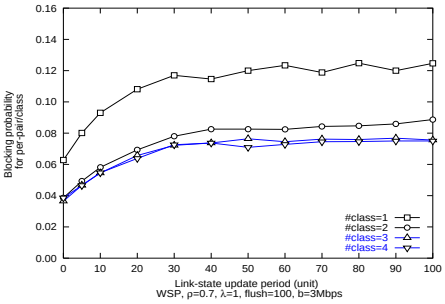

Fig. 8. Blocking probability of Per-pair/class

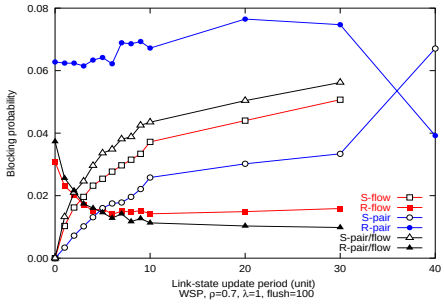

Fig. 9. Routing blocks vs. Signaling blocks

Misleading and Staleness In our architecture, the path cache is used under the link-state update protocol. Both cache-leading and staleness of network state may cause misbehavior of routing schemes. Routing paths extracted from

the cache could be optimistic-leading or pessimistic-leading that we mentioned previously. The extracted path also could be misleading, that is, flows following a misleading path might not find sufficient resources along the path, although there exist alternative paths with abundant resources.

In Fig. 9, we have insight into the blocking probability, blocked either in routing phase (prefix "R-") or in 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 getting larger, more and more flows mistake an infeasible path as feasible path. Therefore, those flows cannot reserve enough bandwidth in the signaling phase and will be blocked. Situation of blocking shifting from routing to 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. As caching mechanism usually does not reflect accurate network state immediately and thus sensitivity of staleness is reduced.

Figure 10 shows the cache misleading probabilities due to caching, i.e.,

$$P_{misl}(\texttt{scheme}) = P_{req}(\texttt{scheme}) - P_{req}(\texttt{per\_flow}), \tag{2}$$

of various routing schemes. In case of no staleness of the network state, since the per-flow routing requires path computation for every single flow, there is no misleading, i.e., P_{misl}(per-flow)=0, regardless of network load and size. Obviously, the per-pair (or per-destination) scheme obtains the highest misleading from the cache. On the other hand, the per-pair/flow and the per-pair/class have little misleading probability. This is because when looking up the P-cache, if the feasibility check fails due to insufficient residual bandwidth along the path held in the P-cache, per-pair/flow and per-pair/class schemes will find another feasible path for the flow. However, feasible path finding could fail due to some paths are nearly exhausted and trunk reservation in [6] can slow down links from being exhausted.

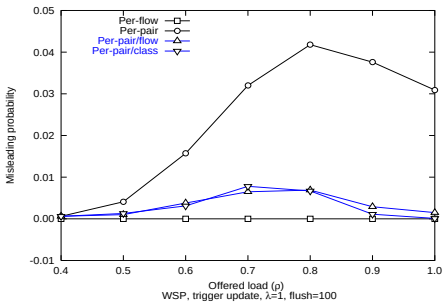

Fig. 10. Cache misleading probability

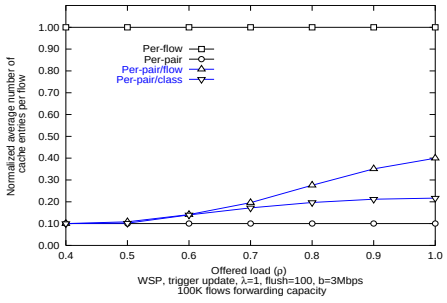

Fig. 11. Average number of cache entries per flow

Cache Size Figure 11 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 the traffic load increases. Statistics in Fig. 11 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 the forwarding capacity of $100K$ flows, $\widetilde{N}_{cache}$(per-pair) is near to 0.1. Similarly, cache of per-pair/class is bounded by $(n-1)^2m$ and has a complexity of $O(n^2m)$, where m is the number of classes. $\widetilde{N}_{cache}$(per-pair/class) is $(n-1)^2m$ divided by the number of forwarding flows.

Finally, $\widetilde{N}_{cache}$(per-flow)=1 in Fig. 11, and thus, the cache size increases dramatically as the number of flows increases in per-flow scheme, which disallows it to scale well for large backbone networks. Compared to the 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% (in light load) to 20–40% (in heavy load) without increasing the blocking probability, compared to the per-flow in Fig. 7.

Number of Path Computations Figure 12 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 are almost the same as WSP. Obviously Fig. 12 and 13 have 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. $flush$).

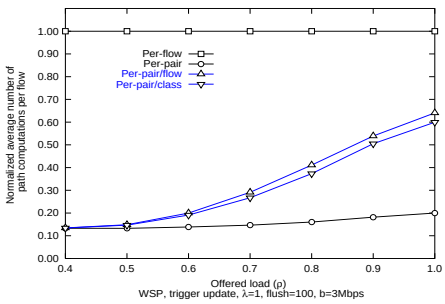

Fig. 12. Average number of path computations per flow

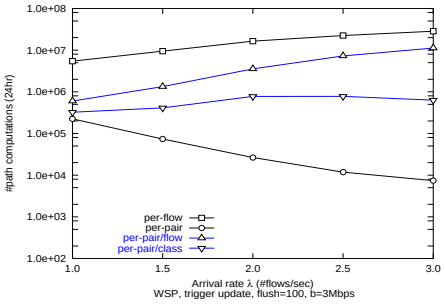

Fig. 13. A snapshot of number of path computations

Figure 13 shows the number of path computations regarding to different flow request rate within a specific twenty-four hours period. Statistics of the per-flow, form the upper bound curve and the per-pair form the lower bound. The per-pair/flow and the per-pair/class schemes are in between, and as the loading increases they either increase and approach the upper bound, or decrease and approach the lower bound. The upward-approaching routing schemes, whose number of path computations is dominated by the number of flow request, including per-flow and per-pair/flow schemes. On the other hand, the number of path computations is dominated by its network size and connectivity, including per-pair and per-pair/class schemes.

Figure 13 reveals that "caching" in the per-pair scheme, reduces number of path computations as the loading increases. This is because there are sufficient flows to relay and survive the life time of cache entry, and a greater percentage of succeeding flow requests do not invoke path-computation. These requests simply look up the cache to make routing decisions. On the other hand, in cases of per-flow granularity, since every flow request requires path computation, the number of path computations increases with the offered load. Notably, there is an *inflection* point in the curve of per-pair/class. Multiple entries in the per-pair/class cache lead to this phenomenon. The property of the basic per-pair cache produces the *concave* plotting while the multiple entries produce the *convex* plotting.

6 Conclusions

This study has investigated how granularity affects the Constraint-based routing in MPLS networks and has proposed 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.

Table 1. Summary of the simulation results

Cache granularity	Compu. overhead	Storage overhead	Blocking	Misleading
Per-pair	$\star\star\star$	$\star\star\star$	$\star$	$\star$
Per-flow	$\star'$	$\star$	$\star\star\star$	$\star\star\star$
Per-pair/flow	$\star\star$	$\star\star$	$\star\star\star$	$\star\star\star$
Per-pair/class	$\star\star\star$	$\star\star\star$	$\star\star\star$	$\star\star\star$

$\star\star\star$: good, $\star\star$:medium, $\star$:poor
$\star'$ can be improved by using path pre-computation.

Extensive simulations are run with various routing granularities and the results are summarized in Table 1. Per-pair cache routing has the worst blocking probability because the coarser granularity limits the accuracy of the network state. The *per-pair/flow* granularity strengthens the path-finding ability just as the per-flow granularity does. Additionally, the *per-pair/class* granularity has small blocking probability with a bounded routing cache. Therefore, this scheme is suitable for the Constraint-based routing in the MPLS networks.

References

1. D. Black, S. Blake, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. *RFC 2475*, December 1998.
2. K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (DS Field) in the ipv4 and ipv6 headers. *RFC 2474*, December 1998.
3. T. Li and Y. Rekhter. A provider architecture for differentiated services and traffic engineering(PASTE). *RFC 2430*, October 1998.
4. Y. Bernet, J. Binder, S. Blake, M. Carlson, S. Keshav, E. Davies, B. Ohlman, D. Verma, Z. Wang, and W. Weiss. A framework for differentiated services. *draft-ietf-diffserv-framework-02.txt*, February 1999.
5. R. Callon, P. Doolan, N. Feldman, A. Fredette, G. Swallow, and A. Viswanathan. A framework for multiprotocol label switching. *draft-ietf-mpls-framework-05.txt*, September 1999.
6. Eric C. Rosen, Arun Viswanathan, and Ross Callon. Multiprotocol label switching architecture. *draft-ietf-mpls-arch-06.txt*, August 1999.
7. Francois Le Faucheur, Liwen Wu, Bruce Davie, Shahram Davari, Pasi Vaananen, Ram Krishnan, and Pierrick Cheval. MPLS support of differentiated services. *draft-ietf-mpls-diff-ext-02.txt*, October 1999.
8. Eric C. Rosen, Yakov Rekhter, Daniel Tappan, Dino Farinacci, Guy Fedorkow, Tony Li, and Alex Conta. MPLS label stack encoding. *draft-ietf-mpls-label-encaps-07.txt*, September 1999.
9. G. Apostolopoulos, R. Guerin, S. Kamat, and S. K. Tripathi. On Reducing the Processing Cost of On-Demand QoS Path Computation. In *Proc. of International Conference on Networking Protocols (ICNP)*, Austin, October 1998.
10. G. Apostolopoulos, D. Williams, S. Kamat, R. Guerin, A. Orda, and T. Przygienda. QoS Routing Mechanisms and OSPF extensions. *RFC 2676*, August 1999.
11. J. A. Shaikh. *Efficient Dynamic Routing in Wide-Area Networks.* PhD thesis, University of Michigan, May 1999.
12. Q. Ma, P. Steenkiste, and H. Zhang. Routing High-Bandwidth Traffic in Max-Min Fair Share Networks. In *SIGCOMM'96*. ACM, August 1996.
13. B. M. Waxman. Routing of Multipoint Connections. *IEEE JSAC*, 6(9):1617–1622, December 1988.
14. D. Zappala, D. Estrin, and S. Shenker. Alternate Path Routing and Pinning for Interdomain Multicast Routing. Technical Report 97-655, USC, 1997.