Admission Control and Path Allocation for SLAs in DiffServ Networks

G. Mardente, M. Mellia, C. Casetti,
Dipartimento di Elettronica – Politecnico di Torino
Corso Duca degli Abruzzi, 24 – I-10129 Torino, Italy
{mardente,mellia,casetti}@mail.tlc.polito.it

***Abstract*— In this paper we consider a Differentiated Service Domain, in which the domain administrator has to decide if to accept or to reject Service Level Agreements (SLAs) requested by users. After introducing the admission criteria which is used to verify if there are enough resources to satisfy the SLA request, we focus our attention to the problem of the SLA routing, i.e., the selection of paths along which traffic may flow. In particular, we show that the construction of an optimal set of paths is equivalent to the construction of a multicast tree, or a *Steiner Tree*, which is know to be an NP-hard problem. We therefore propose a class of simple heuristics, whose performance are assessed by simulations. Results shows that it is possible to increase up to 40% the amount of capacity a network provider can reserve to SLA requests without violating the QoS constraints or to reduce the SLA blocking probability by a order of magnitude by using the proposed algorithms.**

***Index Terms*— DiffServ, Admission Control, Routing, QoS**

I. Introduction

Traffic patterns on today's Internet have become more and more unpredictable, shifting from the ubiquitous client-server paradigm of the early days of the World Wide Web, to the peer-to-peer frenzy of the past few years. As if predicting user traffic were not sufficiently demanding, the introduction of a wide range of mobile services over the Internet is bound to give service providers quite a few headaches too. For these reasons, adequate tools to support the provision of Quality of Service (*QoS*) guarantees to end users are sorely needed. Among them, the Diffserv [1] architecture is frequently touted as the cure-for-all solution. DiffServ requires packet classification at the network ingress, and provides a differentiation of the treatment according to a (small) set of classes, named Per-Hop Behaviors (PHBs). The various PHBs define a rich toolbox for differential packet handling by individual IP routers. In the DiffServ framework, a service contract, or Service Level Agreement (SLA), is established between a customer and a service provider, to specify the forwarding service that a customer should receive. The service contract, though, does not discriminate among packet destinations, (or sources if they are being received by the customer). For this reason, it is of paramount importance that a service provider be capable of predicting, as it were, which parts of its core network are likely to become overloaded as a new SLA signs in, and take appropriate measures. A form of preemptive measure is to route a new SLA over a set of paths that, at the same time, satisfy the user's QoS requirements while making sure that the bandwidth is evenly utilized across the whole domain. This paper addresses the issue of where to route traffic from new SLAs, once they are admitted into a provider's network.

Before tackling the problem of SLA routing, it seems useful to recall the cornerstones of the DiffServ architecture. Common DiffServ PHBs are Expedited Forwarding (EF) [2], Assured Forwarding (AF) [3], and classic Best Effort (BE). The purpose of the EF PHB is to carry traffic from endpoints as if it traveled over a point-to-point connection, or a "virtual leased line"; on it, deterministic QoS guarantees are offered. Provisioning the network for EF traffic is often accomplished by giving it strict priority over traffic marked by other code points. The AF PHB instead offers a soft QoS guarantee: within each AF class, IP packets are marked with one of three possible drop precedence values. In case of congestion within a node, it tries to protect packets with better service profile by preferably discarding those with a higher drop precedence value. Traffic offered by the user is metered at its ingress node according to the user's traffic profile, and packets are marked with the contracted drop precedence value if the data rate is below the contracted rate; otherwise, they are marked with a higher drop precedence value. The AF PHB can be used in a point-to-point setting, as well as in point-to-multipoint configurations, where traffic can flow to different destinations (or come from different sources) at the same time.

The DiffServ QoS paradigm is now supported in a number of IP routers of different make; still, not

This work was supported by the Italian Ministry for University and Scientifi c Research under the project TANGO.

many network operators are yet exploiting DiffServ to offer QoS guarantees to their customers, in spite of the potential increases in revenues. This fact is due in part to the difficulty in setting the parameters of the algorithms used for the differentiation of the treatment offered to the different traffic classes, but even more to the difficulty in determining how to implement an SLA Admission Control (*SLA-AC*) algorithm so as to protect the network from overloads, thus allowing the network to meet the QoS guarantees specified in the SLA contract.

We have proposed and discussed a possible SLA-AC algorithm in [4], where an analytical model was developed to characterize the admissibility of a set of SLAs. Using a statistical approach, it approximately predicts whether the admission of a set of SLAs allows the fulfillment of their QoS requirements. We now build on the previous work by proposing a heuristic algorithm that tries to increase the effectiveness of the admission algorithm through a careful selection of routes from sets of sources to sets of destinations. Those SLAs that were accepted, and for which a route was selected, are "pinned" to the links of the route. Commonly, this is achieved using MPLS [5] inside the core network.

II. Admission Control for SLAs in DiffServ Networks

The problem of admitting SLAs in not unlike standard CAC problems [6], where the following inputs are usually required:

- the (logical) *topology* of a single DiffServ Domain, comprising M nodes and L links; each link l is supposed to reserve capacity C_l to the AF class of service. Each node m can be classified as either *ingress/egress* node if users (i.e., non-specific traffic generators) are connected to that particular node, or *core* node, if it is a pure transit node, i.e., no traffic is either generated or directed to that node;
- the set of *users* that request service from the network; each user is attached to an ingress/egress node, and can request more than one service from the network, i.e., multiple SLAs can refer to the same user;
- a *definition of the SLAs* of interest; in particular, we consider two possible types of SLA: the first refers to traffic generated by a single user and directed to (possibly) multiple destinations, while the second considers traffic going from (possibly) multiple sources to a single user.

Each SLA is described in terms of *assured bandwidth* going to (coming from) a *set of possible destinations (sources)* within the same DiffServ Domain, or, possibly, egress (ingress) nodes connected to other domains. The SLAs define statistical bandwidth guarantees on traffic, i.e., traffic transmitted (received) by the source (destination) subscribing to the SLA; therefore, each SLA is characterized by a probability indicating that soft guarantees are used. In addition, a probabilistic *description of the traffic* associated with every SLA is needed, in terms of fraction of overall traffic going from a user to the set of its destinations, or coming from the set of possible sources to the user.

For the sake of clarity, in the rest of the paper we assume that the SLA will refer to traffic generated from a source node and directed to multiple destinations. Indeed, the extension to the generic case is straightforward.

Given this probabilistic description of the traffic an SLA offers to the network, in [4] we defined a methodology to decide the acceptance or rejection of a new SLA request. The proposed criterion is based on the derivation of the statistical description of the overbooking probability[1] for a new SLA, given the SLAs already admitted.

In this paper we focus our attention to the *route selection problem*, i.e., the identification of the set of links to be used to route traffic from the source toward the destination nodes.

III. Problem statement: Routing of SLAs

The problem of finding the set of paths that will be used to transport traffic from a newly requested SLA is formalized using a graph theory approach. The topology of the DiffServ domain is modeled by a Directed Graph $\mathcal{D} = \{\mathcal{V}, \mathcal{A}\}$ in which $\mathcal{V}$ is the set of vertexes ($|\mathcal{V}| = M$) and $\mathcal{A}$ is the set of directed arcs ($|\mathcal{A}| = 2L$). A vertex represents a node in the topology, while two directed arcs $(i,j),(j,i) \in \mathcal{A}$ between nodes $i,j \in \mathcal{V}$ represent a link. Each arc is weighted by two costs, (μ_{ij}, σ_{ij}), which represent the *average* and *standard deviation* of the traffic which is flowing on arc (i,j).

We assume that an SLA_s is described by

- the source (ingress) node s, to which the user requesting the SLA is obtained;
- the bandwidth requested B_s, to be guaranteed with probability Π_s;
- the probability r_{sd} with which the traffic is directed to a user attached to destination (egress) node d; $\sum_{d \in V} r_{sd} = 1$.

The set of nodes $\mathcal{D}_s = \{d \in V | r_{sd} \neq 0, d \neq s\} \subset V$ includes all destination nodes of SLA_s. The cardinality of $\mathcal{D}_s$ will be indicated by D_s. Then, given a set of already accepted and routed SLAs, it is possible to derive for each arc the cost (μ_{ij}, σ_{ij}).

The routing optimization problem can then be formalized as finding the set of arcs T_s which will be used to

[1] Probability that the traffi c crossing any link of a source-destination path exceeds the link capacity.

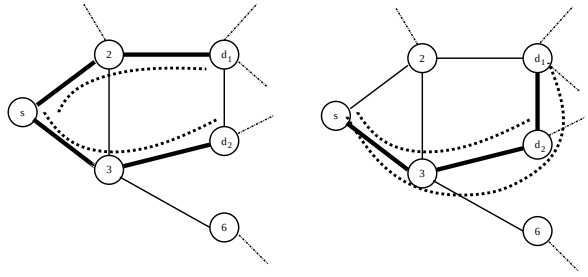

Fig. 1. Two possible trees from source node s to destinations nodes d_1 and d_2 which are obtained by considering $P^i_{s\to d_1} = (s,2),(2,d_1)$, and $P^i_{s\to d_1} = (s,3),(3,d_2),(d_2,d_1)$.

route a new SLA request, so that the maximum average overbooking probability experienced by all $\{SLA_i\}$ over such set of arcs, indicated as $p_s(T_s)$, is minimized, i.e., $\min(\max_i p_i(T_i))$. If $p_s(T_s) \leq \Pi_s$, then the new SLA request will be accepted, otherwise it will be blocked.

The methodology to evaluate the overbooking probability is detailed in [4] and not reported here for lack of space.

This problem is equivalent to the well-known *Steiner Tree problem* [7], which can be summarized as the problem of finding the minimum cost tree T_s that connects a source node s to a subset of vertexes in digraph $\mathcal{D}$. As cost function, the overbooking probability is considered, which unfortunately is a non-linear function of (μ_{ij}, σ_{ij}), transforming the formulation in a non-linear problem. Moreover, also in its original formulation, the Steiner Tree problem is known to be NP-complete, and therefore can only be solved using heuristics.

A. Proposed heuristic

The limited amount of time that can be devoted to solve the problem, i.e., to reply to a user's SLA request, imposes to solve the Steiner Tree problem using heuristics with limited complexity. We therefore propose a simple heuristic, whose complexity is very limited and depends on two tunable parameters. In particular, the construction of the Steiner tree is obtained as union of pre-computed paths, each one connecting the source node s to a particular destination d. An iterative algorithm is then used to compute several trees, and select among them the one which minimizes the maximum overbooking probability $p_i(T_i),\ \forall i$.

Let $\{P^i_{s\to d},\ i = 1,2,\ldots,K\}$ be an ordered set of K precomputed paths joining a source node s to a destination node d. Paths in the set are ordered according to a common metric, e.g., the number of hops, so that path 1 is the shortest path. Let $T_s(n)$ be the solution to the Steiner tree problem obtained at iteration n. For each destination d, a single path, identified as opt, is selected,

```
1.  T_s(0) = ∅, n = 0
2.  // Build the initial solution
3.  forall (d ∈ D_s)
4.  {
5.      T_s(n) = T_s(n) ⋃ {(l,m) ∈ P^1_{s→d}}
6.      opt[d] = 1
7.  }
8.  T*_s = T_s(0)
9.  // Iterate to build other solutions
10. for (n = 1; n ≤ Z; n++)
11. {
12.     forall (d ∈ D_s)
13.     {
14.         for (i = 1; i ≤ K; i++)
15.         {
16.             T_s(n) = ∅
17.             T_s(n) = T_s(n) ⋃ {(l,m) ∈ P^i_{s→d}}
18.             forall (j ∈ D_s \ {d})
19.                 T_s(n) = T_s(n) ⋃ {(l,m) ∈ P^{opt[j]}_{s→j}}
20.             if (max_s(p_s(T_s(n))) < max_s(p_s(T*_s))) then
21.                 T*_s = T_s(n); opt[d] = i
22.         }
23.     }
24. }
25. return T*_s
```

Fig. 2. The Heuristic Algorithm

and $T_s(n)$ is then obtained as the union of all the arcs of $P^{opt}_{s\to d}$. Therefore different solutions are obtained by selecting different sets of paths.

The algorithm we propose tries to efficiently explore the state space of A possible set of paths, whose number grows as a combinatorial function of K. Instead of considering all possible combinations of paths, at each iteration, $K \cdot D_s$ different solutions are tested, each one obtained by simply changing one path at a time, i.e., for a given destination d, test all trees obtained by considering all K paths $P^i_{s\to d}$. This defines $K \cdot D_s$ "neighbors", i.e., solutions that differ from the previous one by a single path. At the end of the iteration, the best neighbor is selected. A maximum number of iterations Z is defined to limit the complexity of the algorithm.

Figure 1 shows two possible trees (arcs included in the tree are thicker), obtained considering the source node s, and two destination nodes d_1, d_2. Two different paths to d_1 are considered, $P^i_{s\to d_1} = (s,2),(2,d_1)$, and $P^j_{s\to d_1} = (s,3),(3,d_2),(d_2,d_1)$, yielding two different trees.

Figure 2 reports a formal description of the algorithm. Lines $1-7$ build the initial solution as the union of all the first-selected paths for all destinations. Lines $10-24$ then iterate Z times the construction of possible better solutions, by building for each destination (line 12) all possible trees considering all K paths from i to d (lines $14-22$). $opt[d]$ is used to store the best path found so far toward destination d.

Considering as basic operation the evaluation of the minimum-maximum overbooking probability for all SLA request, the algorithm complexity is $O(ZMK)$, as at most Z iterations are possible, each of which requires to consider for all destinations $D_s = O(M)$ at most K paths. The computation of K paths is done offline and therefore its complexity does not affect the time required to reply to an SLA request. It can be obtained using variations of the Dijkstra's algorithm with computational complexity $O(KL \log M)$. In particular, in our implementation of the algorithm, the set of paths $\{P^i_{s \to d}\}$ is chosen as the set of K shortest paths from source node s to destination node d.

IV. Performance results

A. Simulation scenarios

We consider a single DiffServ domain, comprising $M = 32$ nodes and $L = 144$ links, arranged in a randomly generated topology[2]. Each link has the same capacity, which is completely devoted to the AF service. 32 users are present, one for each node, all capable of both generating and receiving traffic, so that all nodes are ingress/egress nodes. Each user requests 3 different SLAs, for a grand total of 96 SLAs. To simplify the scenario, we suppose that all SLAs are of the same type. In particular, we consider a scenario where users are traffic sources, and traffic is routed uniformly to D_s destinations, i.e., for each SLA, D_s destination nodes are selected at random for which $r_{s,d} = 1/D_s$, while the remaining $M - D_s - 1$ nodes are not part of the agreement. We present results for $D_s = 31, 16, 8$ possible destinations for each SLA source node.

B. Total Assured bandwidth

As first performance metric, we are interested in the *Total Assured Bandwidth*, AB_{tot}, i.e., the maximum bandwidth the network operator can sell to users without incurring in SLA violation. Given an initial SLA set, we seek a set of committed bandwidths $\{B_i\}$ for which the overbooking probability requests are met, but no B_s can be increased without violating at least one QoS constraint. We call this a *border* configuration, which indicates that the AF network capacity is completely exploited, and no already present SLA can ask for larger bandwidth without violating the guarantees of at least one SLA. The tool we implemented and used in [4] finds border solutions, trying to increase all B_s up to a value in which it is not possible to further increase any of them. Different "increase" algorithms can be applied, so that more than one border point can be found from a starting SLA set. To gauge the impact of the path selection algorithm, each time the tool attempts a bandwidth increase of B_s, a new tree T_s is also generated according to the proposed heuristic. Therefore, more complex tree selection algorithms allow to better exploit network capacity.

[2]Different random topologies were generated using GT-ITM [8] and were tested, without observing major differences from the one presented in this paper.

TABLE I
Percentage of Total Assured Bandwidth considering D_s=31,16,8 destinations (top,medium, bottom plot respectively) for each SLA request.

D_s=31		K			
		1	2	3	4
Z	0	13.13	-	-	-
	1	-	13.97	14.16	14.19
	2	-	14.56	14.87	14.90
	3	-	15.17	15.39	15.41
D_s=16		K			
		1	2	3	4
Z	0	12.74	-	-	-
	1	-	14.89	14.95	15.23
	2	-	15.78	16.11	16.29
	3	-	16.41	16.88	17.10
D_s=8		K			
		1	2	3	4
Z	0	12.41	-	-	-
	1	-	14.55	14.95	15.11
	2	-	15.33	15.80	15.98
	3	-	15.41	15.95	16.10

1) Impact of path set size and number of iterations: To evaluate the impact of the two tunable parameters K and Z which limit the complexity of the heuristic, we report the percentage of the AB_{tot} versus the total network capacity. Table I reports different border points versus different combinations of the Z and K parameters. Three different scenarios are considered, in which $D_s = 31, 16, 8$. The cases $Z = 0$ (or $K = 1$) are all equivalent, because in that cases the algorithm will immediately exit after the initial solution to the Steiner Tree problem is obtained (either no iteration are allowed, or no alternative paths can be considered to build other trees). Therefore, they can be considered as baseline configurations.

Considering the impact of the number of paths K, a limited increase in the percentage of AB_{tot} is observed, while the impact of the number of iterations, Z, has a larger impact. For example, considering $D_s = 31$, the increase of AB_{tot} goes from 13.13% up to 15.41% when $Z = 3, K = 4$, corresponding to an increase of 17%. This is largely due to the increased number of iterations Z rather than to the increased number of paths K. This can be explained by considering that for each increase of Z by a unit, the optimization algorithm explores a different possible solution obtained

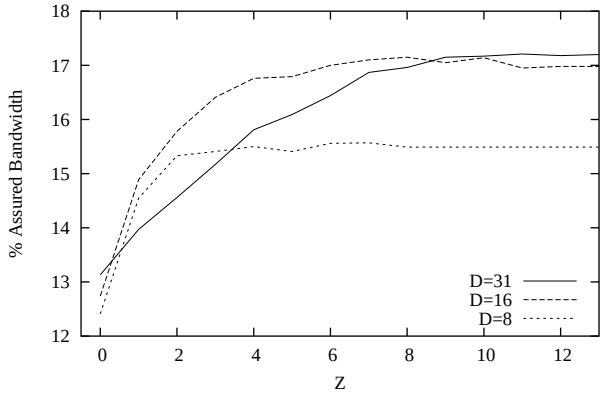

Fig. 3. Total Assured Bandwidth versus Z, considering $K = 2$ and $D_s = 31, 16, 8$.

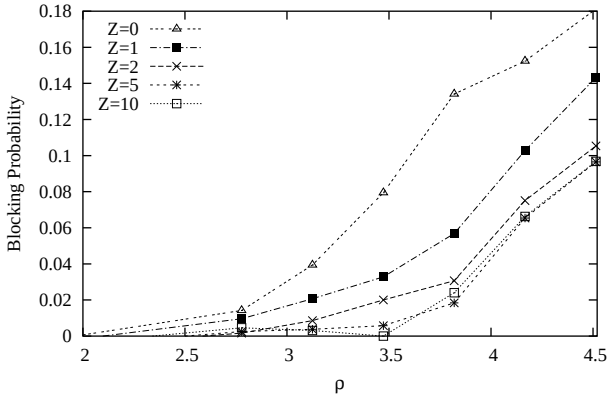

Fig. 4. Blocking Probability of SLA considering $K = 2$ and $D_s = 31$ for different values of Z.

by changing one path to a different destination. On the contrary, increasing the number K of paths generates solutions in which a larger number of paths are tested from each destination, but those paths have lower-quality metrics, (they are longer, in our case) and therefore "consume" more resources when used to route traffic. This holds true for all scenarios, where the gains in the AB_{tot} increase up to 34% ($D_s = 16$) and to 28% ($D_s = 8$). The increased gains are due to a higher degree of variability of scenarios, allowing the optimization algorithm to obtain larger gains.

2) Maximum Number of Iterations Z: The previous results suggest that having just $K = 2$ paths for each pair of source/destination nodes guarantees an increase in Total Assured Bandwidth. It is however interesting to study how large Z can grow before these gains become negligible. Figure 3 gives the answer to the previous question, by reporting the AB_{tot} percentage versus Z for the three previously considered scenarios. As can be observed, all the curves in the plot show an asymptote: for $Z \to \infty$ the AB_{tot} is upper-bounded. In particular, for the $D_s = 31$ scenario, $Z \geq 8$ offers no negligible increase in the Total Assured bandwidth that can be successfully allocated to the AF traffic. For $D_s = 16$, $Z = 6$ is sufficient to reach the maximum gain, while for $D_s = 8$, after $Z = 2$ iterations there is but a negligible gain.

C. Dynamic Scenario - Blocking Probability

While the Total Assured Bandwidth is a performance index that is related to the maximum traffic a network can transport under QoS constraints, another important performance index is the Blocking Probability experienced by users when requesting a service. Smarter allocation algorithms allow network operators to accept a larger number of request, and to increase the revenues. In this subsection, we explore how the proposed algorithm affects the SLA admission control in term of blocking probability. A dynamic scenario is considered, in which users request SLA to the network operator for a limited amount of time. A request is accepted in the network according to the admission policy, otherwise it is refused. SLAs arrive following a Poisson process of parameter λ, and the SLA holding time is exponentially distributed with average duration normalized to 1. Each SLA requests an assured bandwidth B_s = 1Mbit/s toward $D_s = 31$ egress nodes. The same network topology considered in the previous section is considered.

Figure 4 plots the blocking probability versus the offered load to the network, which is defined as $\rho = \frac{B_s}{B_l}\lambda$. The plot shows that a noticeable reduction of the blocking probability is already obtained for $Z = 2$, and a further decrease is achieved for $Z = 5$, while considering $Z = 10$ iterations yields little additional decrease.

This confirms the intuition that it is possible to better allocate network resources with limited complexity without violating the QoS constraints.

D. Computational Complexity

To gauge the computational complexity of the proposed framework, Figure 5 plots the number of seconds required to accept a new SLA requests versus Z and for different values of K. It reports the CPU time used on a 2.4GHz Pentium-IV Linux PC. The time has been averaged over 1000 experiments, each consisting of an SLA request from a random source node s to all destinations ($d = 31$, the worst case). As expected, the CPU time increases linearly with both Z and K, and ranges from about 1s up to 9s, therefore remaining limited and compatible with the reply time of an SLA request.

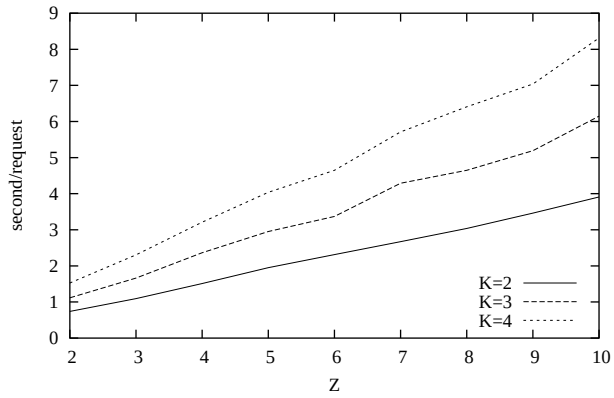

Fig. 5. Time spent to accept an SLA request versus Z and versus K.

V. CONCLUSIONS

The paper proposed a low-complexity, heuristic algorithm to improve the resource selection that allow a service provider to increase the amount of bandwidth it can sell to its users without violating traffic guarantees. Results have shown that, although the solution is non-optimal, it yields remarkable gains after a few iterations.

REFERENCES

[1] D.Black, S.Blake, M. Carlson, E. Davies, Z. Wang, W. Weiss, *An Architecture for Differentiated Services*, RFC 2475, December 1998.

[2] V. Jacobson, K. Nichols, K. Poduri, *An Expedited Forwarding PHB*, RFC 2598, June 1999.

[3] J.Heinanen, F.Baker, W. Weiss, J. Wroclawski, *Assured Forwarding PHB Group*, RFC 2597, June 1999.

[4] M.Mellia, C.Casetti, G.Mardente, M.Ajmone Marsan, "An Analytical Framework for SLA Admission Control in a DiffServ Domain", *IEEE Globecom 2002*, Taipei, TW, November 2002.

[5] E. Rosen, A. Viswanathan, R. Callon, *Multiprotocol Label Switching Architecture*, RFC 3031, January 2001.

[6] Harry G. Perros and Khaled M. Elsayed, "Call Admission Control Schemes: A Review", *IEEE Communications Magazine*, N.34, pp:82-91, November 1996.

[7] P.Winter, "Steiner Problem in Networks: A Survey", *Networks*, Vol.17, pp.129-167, Summer 1987.

[8] GT-ITM: Georgia Tech Internetwork Topology Models, http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html