Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computer Communications 27 (2004) 1330–1340
computer communications
www.elsevier.com/locate/comcom

A scalable architecture for end-to-end QoS provisioning

Spiridon Bakiras*, Victor O.K. Li

Department of Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam Road, Hong Kong, China

Received 23 September 2003; revised 18 March 2004; accepted 13 April 2004

Available online 4 May 2004

Abstract

The Differentiated Services (DiffServ) architecture has been proposed by the Internet Engineering Task Force as a scalable solution for providing end-to-end Quality of Service (QoS) guarantees over the Internet. While the scalability of the data plane emerges from the definition of only a small number of different service classes, the issue of a scalable control plane is still an open research problem. The initial proposal was to use a centralized agent, called Bandwidth Broker, to manage the resources within each DiffServ domain and make local admission control decisions. In this article, we propose an alternative decentralized approach, which increases significantly the scalability of both the data and control planes. We discuss in detail all the different aspects of the architecture, and indicate how to provide end-to-end QoS support for both unicast and multicast flows. Furthermore, we introduce a simple traffic engineering mechanism, which enables the more efficient utilization of the network resources.

Keywords: Admission control; Differentiated services; Quality of service; Resource management; Traffic engineering

1. Introduction

In the past few years, the dramatic increase in the capacity of the Internet core, and the development of powerful compression techniques, have allowed the deployment of new applications such as Internet telephony, video-conferencing, streaming audio/video, etc. These applications are called real-time, since they require the periodic and timely delivery of the content from the source to the destination. Clearly, the traditional best-effort service that is provided in the current Internet cannot offer an acceptable level of service quality to this type of applications. To address this problem, the Internet Engineering Task Force (IETF) has proposed the Differentiated Services (DiffServ) architecture [1] as a scalable solution for providing end-to-end Quality of Service (QoS) guarantees over the Internet. The scalability issue is of outmost importance, since, in the future, the number of flows that will require some QoS guarantees is expected to be very large. Consequently, a core router should be able to accommodate thousands of QoS-sensitive flows at any time instant.

The basic idea of the DiffServ architecture is that only edge routers should manage traffic on a per flow basis. Core routers should not keep any kind of per flow state, and should process traffic on a much coarser granularity. At the data plane this goal is achieved by specifying different Per Hop Behaviors (PHBs), where packets belonging to the same PHB form a Behavior Aggregate (BA) and receive identical service at the core routers. Specifically, the edge routers will be equipped with flow classifiers, policers, and markers that will properly mark the incoming packets by setting a number of bits on the DiffServ Codepoint (DSCP) [2] field of the IP packet header. The DSCP value will indicate the corresponding PHB, and the core routers will forward the packets based on their DSCP value (by utilizing several scheduling and buffer management techniques).

The IETF has currently specified two different PHBs. The *Expedited Forwarding* (EF) PHB [3] offers the equivalent of a leased line (i.e. low delay, loss, and jitter) between a source and a destination. This is accomplished by giving EF traffic strict priority over the traditional best-effort traffic inside the DiffServ domain. However, each flow has to specify in advance the required bandwidth so that the appropriate resources may be reserved inside the network. In addition, the maximum burst size that is allowed is equal to two Maximum Transmission Units (MTUs).

* Corresponding author. Tel.: +852-2857-8487; fax: +852-2559-8738.
E-mail addresses: sbakiras@eee.hku.hk (S. Bakiras); vli@eee.hku.hk (V.O.K. Li).

0140-3664/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2004.04.002

The edge routers will police each flow, and the non-conforming packets will either be dropped or shaped. The *Assured Forwarding* (AF) PHB group [4] does not offer hard QoS guarantees, but instead defines four different AF classes with three different levels of drop precedence within each class. Each AF class is assigned a certain amount of bandwidth at each node, and when the amount of traffic exceeds this bandwidth, packets will be dropped according to their drop precedence value.

While the scalability of the data plane emerges from the definition of only a small number of PHBs, the issue of a scalable control plane is still an open research problem. The initial proposal was to use a centralized agent, called Bandwidth Broker (BB) [5], to manage the resources within each DiffServ domain and make local admission control decisions. The centralized approach removes the burden of admission control from the core routers, but there might be some scalability considerations if the BB has to process thousands of requests per second. Moreover, this approach has certain disadvantages that are inherent to any centralized architecture.

- The links around the BB will become very congested when the traffic load from the signaling messages is high.
- The BB must maintain per flow information about every flow that is currently active inside its domain.
- The BB is a single point of failure (i.e. undesirable in reliability considerations).

In this article, we propose an alternative decentralized architecture, where the local admission decisions are made independently at the edge routers of each domain. The BB in each domain is only responsible for periodically updating the allocation of the resources inside the domain, according to some measurements of the traffic load at the edge routers. We discuss in detail all the aspects of the proposed architecture (i.e. intra- and inter-domain routing, admission control, packet forwarding, etc.), and indicate how to provide end-to-end QoS support for both unicast and multicast flows. Furthermore, we introduce a simple traffic engineering mechanism, which enables the more efficient utilization of the network resources.

The remainder of this article is organized as follows. In Section 2 some related work on DiffServ resource management is presented. In Section 3 we give the details of the proposed architecture, and also discuss various implementation issues. In Section 4 the results of the simulation experiments are presented, while Section 5 concludes our work.

2. Related work

The standardization of the DiffServ architecture by the IETF triggered the initiation of several projects, which aim to provide DiffServ-based QoS guarantees over the Internet. The largest of these projects is the Internet2 project, which involves over 200 universities, corporations, and other organizations worldwide. The main objective of the Internet2 QBone initiative [6] is to build an experimental testbed for providing end-to-end QoS guarantees in a scalable manner. Their approach on resource management follows the initial proposal of a centralized BB, which is responsible for managing the resources within a DiffServ domain, and performing intra-domain admission control. For end-to-end resource reservations, inter-BB signaling is required between the BBs of adjacent domains.

One direction towards improving the scalability of the resource management is based on aggregated resource reservations between DiffServ domains. The BB is still the centralized agent responsible for resource reservation, but the scalability is improved by reserving resources for aggregate traffic between different domains. In Ref. [7] a two-tier model is introduced, where each domain is assumed to have long-term bilateral agreements with each of its neighbors, specifying the amount of traffic that will be exchanged between them. Whenever there is an increase in the traffic between two domains, the BBs will re-negotiate and make new agreements. In Ref. [8] a Clearing House architecture is proposed, where multiple basic domains are clustered to form a logical domain. In this way, a hierarchical tree is created, where the BB of the logical domain is responsible for resource reservation across the basic domains. The BBs at the basic domains forward only aggregation of inter-domain requests to the BB of the logical domain, thus enhancing the scalability of this architecture.

Alternatively, an approach based on the Multiprotocol Label Switching (MPLS) [9] architecture has also been considered in Refs. [10,11]. In these two architectures, reservations for aggregate traffic are made between pairs of edge routers on specific Label Switched Paths (LSPs) inside the domain. All the QoS-sensitive flows will then follow the appropriate LSPs, in order to receive the requested QoS. The work in Ref. [11] is introduced as part of the Traffic Engineering for Quality of Service in the Internet at Large Scale (TEQUILA) project.

3. An architecture for end-to-end QoS provisioning

In this section we introduce an architecture for DiffServ-based networks, which enhances the scalability of both the data and control planes. The goal is to push most of the functionality to the edge of the network, and maintain a simple core, which only performs a standard packet forwarding function. Our assumption is that the Internet consists of several independently administered DiffServ domains that are interconnected in order to provide global connectivity. One typical example is shown in Fig. 1, where the sender (S) and the two receivers ($R1$ and $R2$) are interconnected through three different domains.

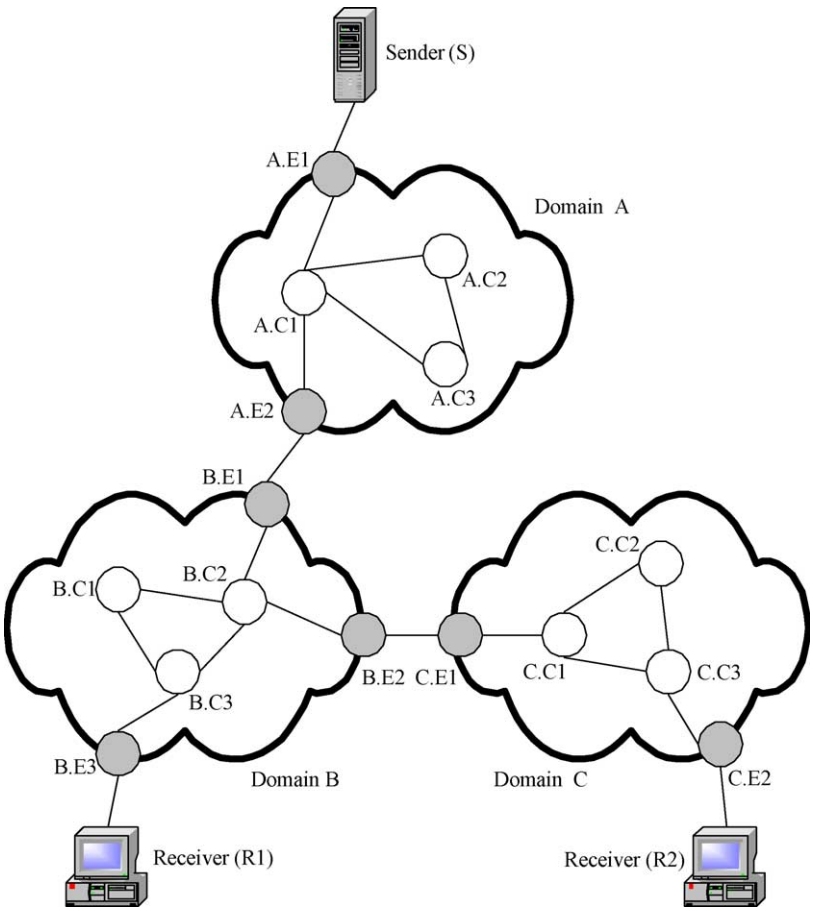

Fig. 1. The Differentiated Services architecture.

Each DiffServ domain consists of a BB (not shown in the figure), and the core and edge routers. The BB will periodically exchange control messages with the edge routers for the purpose of resource management.

3.1. Intra-domain routing

Routing is the process of correctly identifying the next hop at each node (router) so that a packet will be able to reach its final destination. In this work we focus on intra-domain routing, and assume that all the DiffServ domains use a standard inter-domain routing protocol, such as the Border Gateway Protocol (BGP) [12], to exchange reachability information with their neighbor domains. All the edge routers in each domain will participate in this information exchange.

Routing with QoS guarantees requires the reservation of enough resources along the path from the source to the destination. Therefore, unlike best-effort traffic routing, a path has to be established in advance between the source and destination nodes, and all the packets should follow the same path. In our architecture, we adopt a source routing scheme, where the ingress router of the domain will identify the complete path towards an egress router. Specifically, during the initialization of the network, the BB will pre-compute k different paths to carry the traffic between each pair of edge routers, and it will distribute this information to all the routers in its domain. In the simulation experiments presented in Section 4, we used the well known k-shortest path algorithm [13] for the path selection, where the hop count was used as the link metric.

The source routing approach was adopted for several reasons: (i) it facilitates fast packet forwarding which will be further discussed in Section 3.2, (ii) it follows the general principles of the DiffServ architecture, by completely isolating the core routers from the admission control procedure, and (iii) it provides the means for implementing traffic engineering mechanisms inside the domain.

Finally, we assume that a standard link state routing protocol, such as OSPF, operates inside each domain, in order for the edge routers to advertise their routes to other networks, and also to exchange information with all the core routers regarding link or node failures. In other words, our intra-domain routing protocol will operate on top of any link state protocol, and the ingress routers of the domain will use the existing link state database to identify the corresponding egress router where a packet should be forwarded to.

3.2. Packet forwarding with IPv6

The slowest process in the forwarding path of an IP router is the multi-field classification and routing procedure. When a packet is received at a router, the next hop is decided by looking into several fields on the IP header (e.g. IP addresses, TCP/UDP port numbers, etc.), and then finding the appropriate entry at the local routing table. This operation will be even more complicated for QoS-sensitive flows, since their packets should follow exactly the same path. Clearly, this procedure will become the bottleneck in a multi-Gbps router.

The IPv6 packet header contains a new 20-bit field, which does not exist in the earlier IPv4, called *flow label*. Using this field in the context of a source routing architecture, enables us to increase considerably the speed of the forwarding path. As we mentioned earlier, for each pair of edge routers inside a domain, there will be k pre-computed paths connecting them. We may then assign one flow label value to each one of these paths, and construct new (much smaller) routing tables inside the core of the domain, based only on flow labels. We should emphasize that the flow label in our approach is not related to the traditional definition of a flow (i.e. a connection between a certain source–destination pair). Instead, we use the flow label field in the IP header, in order to identify a unique path within an AS domain. As a result, any path within a domain will be assigned a specific flow label value, and all the packets (from any packet flow) that have to follow this path will be marked with that exact flow label value. Therefore, the unicast routing table within a domain will be static, and its maximum size will be equal to the total number of paths that we choose to identify. Furthermore, during the resource reservation procedure, the ingress router will select one of the k paths for each new flow, and then mark all the packets that belong to this flow with the corresponding flow label value.

An alternative method would be to use the routing header option in IPv6, since it provides exactly the same functionality (i.e. source routing). The reason why we did not choose this option is due to the large overhead that it introduces. Each IPv6 address entry is 16 bytes, and for a multi-hop path this approach could increase substantially the protocol overhead (especially for small packet sizes). In addition, using the routing header option does not help in the case of multicast communication. In the following paragraphs we will indicate how the flow label approach may be exploited to facilitate multicast routing.

Notice, that our forwarding scheme is similar to the MPLS architecture, but it does not require inter-router signaling as in the case of the label distribution protocol (LDP) in MPLS networks. The BB will be the centralized agent responsible for distributing the routing tables to the domain routers. In particular, after the BB has selected the k paths for each pair of edge routers, it will send the appropriate routing table entries to all the routers in the domain. These entries will be in the form of $\langle flow\_id, link\_id \rangle$, where *link_id* indicates the outgoing interface where the packet should be forwarded to. Having a centralized agent distribute all the routing information does not affect the scalability of the architecture, since this information will be distributed only once (at the initialization of the network) and will be updated only if a new link or router is introduced in the topology.

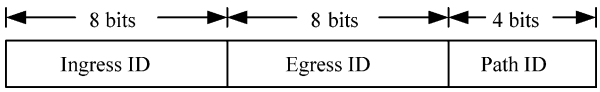

Fig. 2. Flow label assignment.

One example of how to assign flow labels to the different paths inside a domain is depicted in Fig. 2. With this assignment, for instance, we are able to identify a maximum of 16 different paths between any pair of 256 edge routers. To further illustrate the concept of flow-based routing, consider the edge routers $B.E1$ and $B.E3$ in domain B (Fig. 1). Assume that the ID of $B.E1$ is 0, and the ID of $B.E3$ is 2. There are exactly two paths connecting these routers (through $B.C2 \rightarrow B.C3$ or $B.C2 \rightarrow B.C1 \rightarrow B.C3$), and suppose we assign them the IDs 0 and 1, respectively. These two paths may then be represented by the flow label values '0.2.0' and '0.2.1'. As a result, if $B.C2$ receives a packet with a flow label value 0.2.1, it will forward it towards $B.C1$ and not towards the shortest path (i.e. through $B.C3$).

For multicast communication, packet forwarding is slightly more complicated, but we may still utilize the above flow-based routing mechanism to maintain a scalable forwarding path. Let us consider Fig. 1 again, and assume that receivers $R1$ and $R2$ wish to join a multicast group created by the sender S in domain A. Suppose node $R1$ joins the multicast group first. Then, $B.E1$ will send a control message to the core routers $B.C2$ and $B.C3$ in the form of $\langle INSERT, MG, 0.2.0 \rangle$, or else "if you see a packet destination address MG, use flow label 0.2.0 to forward it". This message will create an entry in a local multicast routing table inside the core routers. When $R2$ joins the group, the message $\langle INSERT, MG, 0.1.0 \rangle$ will be sent to $B.C2$, forcing it to forward the packets towards both $B.C3$ and $B.E2$. If a node decides to leave the multicast group, similar *DELETE* messages will be sent, if necessary.

3.3. End-to-end admission control

Resource reservation is an essential part of any network that provides QoS guarantees, and an appropriate signaling protocol is necessary in order to perform this function. In our architecture, the receiver nodes will initiate the signaling procedure for the resource reservation, while the intermediate ingress routers will be responsible for admitting or rejecting the reservation requests. In the following paragraphs we illustrate how admission control may be

performed across multiple DiffServ domains for the case of unicast and multicast flows.

Let us consider unicast flows first, and assume that $R1$ (Fig. 1) wishes to receive some QoS-sensitive data from the sender S at domain A. Then, the end-to-end admission control will be performed as follows (with an RSVP-like signaling protocol).

(1) $R1$ will send a *PATH* message towards S, indicating the required amount of bandwidth b.
(2) The *PATH* message will reach $B.E1$ which will be the ingress router for that particular flow. Therefore, it will check whether there are enough resources to carry this flow towards $B.E3$. The details of the admission control decision will be discussed in Section 3.4, where we introduce the traffic engineering mechanism.
(3) If there are not any sufficient resources, the request will be rejected. Otherwise one of the k available paths towards $B.E3$ will be selected, and the *PATH* message will be forwarded towards S.
(4) The *PATH* message will reach $A.E1$ which will also perform the admission control as in steps (2) and (3).
(5) If this request can be accommodated, $A.E1$ will forward the *PATH* message to the source node S.
(6) If S wishes to establish this connection, it will send the *RESV* message back to $R1$.
(7) While the *RESV* message travels back to the destination node, all the intermediate edge routers will configure their traffic shapers, policers, and markers to account for the new connection.

The signaling procedure for multicast flows is essentially identical to the one described above, with only a few minor additions. We assume that a multicast group is identified by the pair (S, MG), i.e. the IP address of both the source and the multicast group. For multicast groups where any receiver can also be a sender, we assume that only one node is allowed to create the multicast group, and this node will be the designated source. This is a very reasonable assumption, which greatly simplifies the routing of the signaling messages when new nodes join a multicast group. Going back to our example, let us consider the case where node $R1$ joins the multicast group initiated by node S. The signaling message flow will then be as follows.

(1) $R1$ will send a *PATH* (or join) message towards S, indicating the required amount of bandwidth b.
(2) If $R1$ may also be a sender (i.e. in multipoint-to-multipoint communication), $B.E3$ will reserve the appropriate resources towards $B.E1$, and forward the *PATH* message towards S.
(3) Steps (2)–(7) from the unicast case will be performed in the same manner. In the case of multipoint-to-multipoint communication, both the ingress and egress routers will perform admission control, since the multicast data may flow in both directions.
(4) $A.E1$, $A.E2$, $B.E1$ and $B.E3$ will send the appropriate control messages to the core routers of their domains, in order to set the corresponding entries in the multicast routing tables (as described in Section 3.2).

If $R2$ decides to join the multicast group, the same steps as above will take place. However, as soon as $B.E1$ receives the *PATH* message, it will not forward it to S, since it already has a reservation entry for that particular multicast address. Instead, it will send the *RESV* message back to $R2$, given that there are enough resources to carry the multicast traffic towards $B.E2$.

Notice, that in the multipoint-to-multipoint scenario we have assumed that only one member is allowed to send packets at any given time. If this is not the case (i.e. if all members are allowed to transmit simultaneously), then the *PATH* messages will have to travel always back to the initiator of the group (node S in our case), in order to reserve additional bandwidth for each new member. Also, we have made the implicit assumption that the ingress routers corresponding to the initiator of the group will act as the core nodes of the multicast tree inside their own domain. In our example, since node S initiated the multicast group, $A.E1$ will be the core of the tree in domain A, $B.E1$ will be the core in domain B, and $C.E1$ will be the core in domain C. In other words, if $R1$ wishes to send a packet to the multicast group, this packet will be forwarded towards $B.E1$, and each router along the path will forward it to all the corresponding interfaces (based on the local multicast routing table), except the one that the packet was received from. Clearly, this is not the optimal way to distribute the multicast traffic, but it simplifies greatly the multicast routing protocol, since it avoids the construction of a separate multicast tree for each member of the group.

It is evident that our model is using the per-flow reservation paradigm, which is the basis of the IntServ architecture. However, our approach overcomes the scalability problem of IntServ, by (i) eliminating the core routers from the resource reservation procedure, and (ii) decentralizing the resource reservation procedure. Moreover, the resource reservation phase is completed by the exchange of only two messages. The first message (*PATH*) is sent from the receiver towards the sender, and is followed by the reply message (*RESV* or reject) which is sent back to the receiver. These messages are only intercepted by the corresponding edge routers (i.e. ingress and egress) of each domain, and are not processed at all by any of the core routers. The advantages of this decentralized architecture may be summarized as follows.

- The signaling overhead for connection set up is spread across multiple links, avoiding the congestion of the links around the centralized BB.
- The BB does not need to maintain per flow information about every flow that is currently active inside its domain. This information will be distributed across the edge routers of the domain.

- It offers a scalable solution for dynamic connection set up with very fast response time (equal to the round-trip delay between the source and the destination).
- A temporary failure of the BB does not affect the functionality of the domain, since the admission control may be performed based on the current resource allocation vector. The failure of an edge router has no effect, since no traffic should be routed through this node.
- The core routers are not involved at all in the resource reservation procedure.
- The edge routers are able to dynamically route the best-effort traffic through paths that are possibly not congested, i.e. paths for which the edge router has reserved only a small portion of the allocated bandwidth.

Before we continue, let us discuss briefly the amount of state information that needs to be stored in the different routers of each domain. First, the ingress routers will maintain per flow state about all the active QoS flows for which they have an entry in the reservation database. Specifically, they will keep the following pieces of information: (i) the source and destination IP address and port number, (ii) the corresponding egress router IP address, (iii) the assigned flow label value, and (iv) the amount of bandwidth reserved.

The egress routers between two domains (e.g. nodes $A.E2$ and $B.E2$ in Fig. 1) do not need to keep any per flow state. Instead, they will keep some information about the aggregate amount of QoS traffic that leaves the domain through that particular node. This information is important, since the egress routers will need to shape the QoS traffic so that it does not exceed the total amount of reserved bandwidth towards other domains.

Finally, the egress routers located near the clients (i.e. $B.E3$ and $C.E2$), which are often referred to as leaf routers, will also need to maintain per flow state about all the clients that are directly connected to them. This is not required for the provisioning of QoS guarantees, but it is necessary in order to implement *soft state* resource reservation. In particular, the receiver nodes will periodically send *refresh* messages to their local leaf router, which will verify the validity of the reservation. If the leaf router does not receive a refresh message within a certain timeout period, it will send a message to the corresponding ingress router to release the necessary resources.

3.4. Traffic engineering

Traffic engineering is the process where the routes between any two endpoints of a certain domain are dynamically selected, according to the current traffic load, so as to optimize the utilization of the network resources. In our architecture, the BB will be responsible for periodically updating the allocation of the resources inside the domain, according to some measurements of the traffic load at the edge routers. When the allocation of resources is completed, all the edge routers will be able to make instantaneous and independent admission control decisions for new connection requests.

Let us consider the following model for a generic DiffServ domain.

- There are M edge routers in the domain.
- Every edge router will maintain the exact information about the network topology of the domain.
- There are $N = M(M-1)$ different router pairs to which we shall refer to as Source-Destination (SD) pairs.
- For each SD pair (i,j) there are k pre-computed paths.

The BB will periodically (e.g. every T minutes) allocate a certain amount of bandwidth on every link of the network topology to each one of the M edge routers. This information will be distributed to the corresponding edge routers, and will enable them to make independent admission control decisions. In other words, each router will see the same network topology, but with different link capacities, which may be used entirely by itself. If a new reservation request arrives for SD pair (i,j), the ingress router i will select the shortest of the k available paths which can satisfy the request. A similar shortest path tree method may be used for the case of multicast flows.

In order for the BB to allocate the resources in a more efficient manner, it will need some information regarding the traffic load requirements between the different SD pairs. We assume that the source node of each SD pair (i,j) will keep track of the following variables in order to measure the traffic load towards the destination j :

- Total number of requests, r_{ij}.
- Total amount of bandwidth requested, b_{ij}.
- Total holding time for completed connections, t_{ij}.
- Total number of completed connections, c_{ij}.

Whenever a new request arrives at a certain node i, the corresponding r_{ij} will be incremented by one, while the requested bandwidth b will be added to b_{ij}. If a connection is terminated, its holding time will be added to t_{ij}, and c_{ij} will be incremented by one. Every time interval T, the BB will poll every edge router i, and it will ask for the following two pieces of information: (i) the current amount of resources reserved on each link (l,m), $\{x_i(l,m)\}$, and (ii) the required amount of additional bandwidth w_{ij} (if any) towards each destination j. After receiving all the information, the BB will run an algorithm which will calculate the new values of $\{x_i(l,m)\}$.

For the calculation of w_{ij} we may model each SD pair as an M/M/m/m queueing system. We assume that new requests arrive according to a Poisson process with rate λ, and that the system can accommodate up to m concurrent connections. The service time for each connection is exponentially distributed with mean $1/\mu$. Using Erlang's B formula, we may calculate the blocking probability for

new requests as follows

$$p_m = \frac{(\lambda/\mu)^m/m!}{\sum_{k=0}^{m}(\lambda/\mu)^k/k!}$$

In our case, $\lambda = r_{ij}/T$ and $1/\mu = t_{ij}/c_{ij}$. Then, we can calculate the smallest value of m for which the blocking probability is less than, for example, 1%. Therefore, the total traffic load for SD pair (i,j) will be equal to mb_{avg}, where $b_{\text{avg}} = b_{ij}/r_{ij}$ is the average amount of requested bandwidth. Assuming that the current amount of reserved resources for SD pair (i,j) is s_{ij}, w_{ij} will be given by $w_{ij} = \max\{0, mb_{\text{avg}} - s_{ij}\}$.

We understand that the M/M/m/m model is not entirely accurate, but we believe that it provides a very good approximation for predicting the amount of required bandwidth between a pair of edge routers. For example, it is generally acknowledged that the arrival of new requests from a large population size (e.g. for voice or video) may be modeled as a Poisson process. Also, the holding times for voice calls have long been considered as exponential. For video calls (e.g. video-conference, streaming video) the exponential assumption is not so accurate, but in the general case the holding times will consist of many short calls (e.g. a short video clip, a videophone application) with only a small percentage of longer calls (e.g. a two-hour movie). In any case, the exponential holding time assumption provides a good approximation and, most importantly, allows us to use a simple formula for calculating the amount of required resources. Using a more elaborate queueing model would increase the complexity of the traffic engineering module, while the return in terms of accuracy would be uncertain.

For the resource reallocation procedure we use a simple greedy algorithm (Fig. 3) which may be summarized as follows. We add, in a round-robin manner, a very small amount of bandwidth dx to each SD pair with $w_{ij} > 0$. This amount of bandwidth will be added to the path n which minimizes a certain cost function. The above procedure will continue until all SD pairs have either $w_{ij} \leq 0$ or have become *saturated* (i.e. no more resources can be allocated to them). The cost function that we used was the average number of packets in an M/M/1 system [14], given by

$$\text{cost}_n = \sum_{(l,m)} \frac{F_{lm}}{C_{lm} - F_{lm}}$$

where (l,m) are the links that belong to path n, F_{lm} is the flow on link (l,m), and C_{lm} is the capacity of link (l,m). This is a very efficient cost function for spreading the traffic load among different paths.

After this procedure is terminated there might be some resources left that have not been allocated to any path. These unused resources will be allocated equally among all the non-saturated SD pairs, in a manner similar to the one described above. We may easily modify this algorithm to account for other resource allocation policies as well.

```
ResourceReallocation(s[i].x[l][m], w[i][j])
    /* Allocate necessary resources */
    while (exists non-saturated SD pair (i,j) with w[i][j] > 0)
      for (all SD pairs (i,j))
        if (w[i][j] > 0 and (i,j) not saturated)
          for (n = 0; n < k; n++)
            calculate cost[n] for additional bandwidth dx;
          if (cost[n]==∞, for all n)
            SD pair (i,j) is saturated;
          else
            select path n, where cost[n] = min;
            w[i][j] -= dx;
            for (all links (l,m) on path n)
              s[i].x[l][m] += dx;

    /* Allocate unused resources */
    while (exists non-saturated SD pair (i,j))
      for (all SD pairs (i,j))
        if ((i,j) not saturated)
          for (n = 0; n < k; n++)
            calculate cost[n] for additional bandwidth dx;
          if (cost[n]==∞, for all n)
            SD pair (i,j) is saturated;
          else
            select path n, where cost[n] = min;
            for (all links (l,m) on path n)
              s[i].x[l][m] += dx;

    return s[i].x[l][m];
```

Fig. 3. The resource reallocation algorithm.

For example, two neighbor domains may choose to limit the amount of traffic that is exchanged between them. This may be realized by limiting the capacity of the inter-domain link in the resource management algorithm. Another example is the more permanent connection set up of a Virtual Private Network (VPN). This may also be realized by setting up these connections in advance, and then limiting the capacity of the corresponding links.

Finally, we should note that our architecture is easily applied to the Relative DiffServ model as well. Instead of calculating the values of $x_i(l,m)$ for the EF class only, different values may be assigned for other service classes as well. The edge routers will then be able to reserve resources for any service class requested by the end-user. In other words, each additional service class may be treated (from the resource management point of view) exactly as the EF class. The relative service (e.g. delay, bandwidth) received by each class will be determined by the packet scheduling algorithm that is implemented at the routers, and is beyond the scope of this work.

4. Simulation results

We simulated the proposed resource allocation algorithm in the MCI Internet topology of Fig. 4, which has been widely used in many studies [15]. We assume that all the links have a capacity of 2.4 Gbps, and that all the capacity may be allocated to QoS flows. In a real network, though,

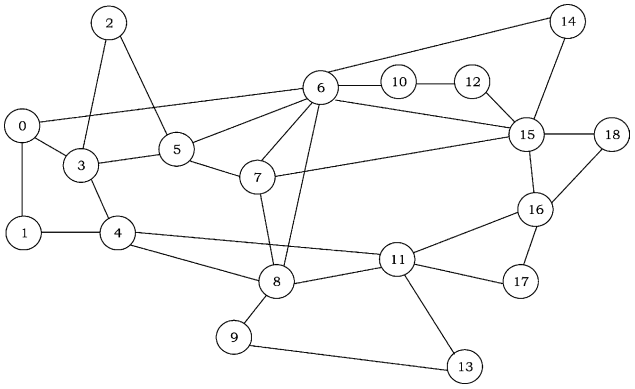

Fig. 4. The simulated network topology.

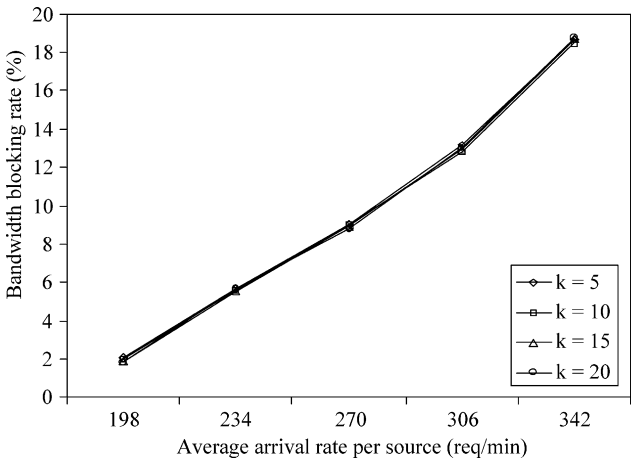

Fig. 5. Effect of the number of paths k on the bandwidth blocking rate (unicast flows).

the service provider will allocate a portion of the available bandwidth to QoS-sensitive traffic, according to some policy. This topology represents an autonomous DiffServ domain, so we have only simulated the intra-domain resource management scheme. As we have shown in Section 3.3, inter-domain admission control is performed as a series of independent intra-domain admission decisions.

New requests may arrive at any one of the 19 routers, which means that there are $N = 342$ different SD pairs. The new requests arrive at each node i, according to a Poisson process with mean λ_i. The arrival rates for the different nodes are chosen from a uniformly distributed random variable in the interval $[1, \lambda_{\max}]$. The value of $\lambda_{\max}$ is properly adjusted in order to control the average arrival rate per node. For unicast flows, we randomly select one node as the destination, while for multicast flows a fixed number of destinations are selected (either 6 or 9). The duration of each connection is exponentially distributed with mean 20 min, and the required bandwidth is uniformly distributed in the interval [100, 1000] kbps. In each experiment we simulate 24 h of real time, and collect the results after an initial warm-up period. The bandwidth blocking rate is used as the performance metric, i.e. the percentage of actual bandwidth that is rejected due to insufficient resources inside the domain.

In the first experiment we investigate the impact of the number of paths k that are pre-computed for each SD pair. The results are depicted in Figs. 5 and 6, where we may observe that setting up more paths per SD pair will generally result in better performance only in the case of multicast flows. This result is anticipated, since having a lot of available paths is important when constructing a shortest path tree. For unicast flows, the performance is not affected by the number of paths, since it is always better to route traffic through shorter paths. Therefore, adding more paths, which are normally longer, does not reduce the bandwidth blocking rate.

We then investigate the impact of the reallocation interval T on the performance of the resource allocation algorithm. The 'static' curve in Figs. 7 and 8 corresponds to a system where the resource allocation is manually configured according to some long term traffic measurements. In the simulation experiments, this curve is produced

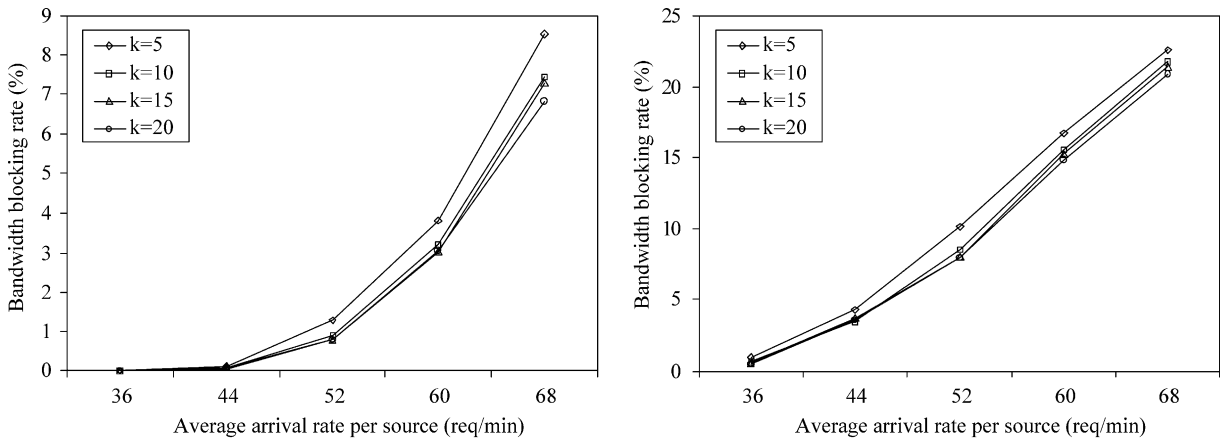

Fig. 6. Effect of the number of paths k on the bandwidth blocking rate (multicast flows).

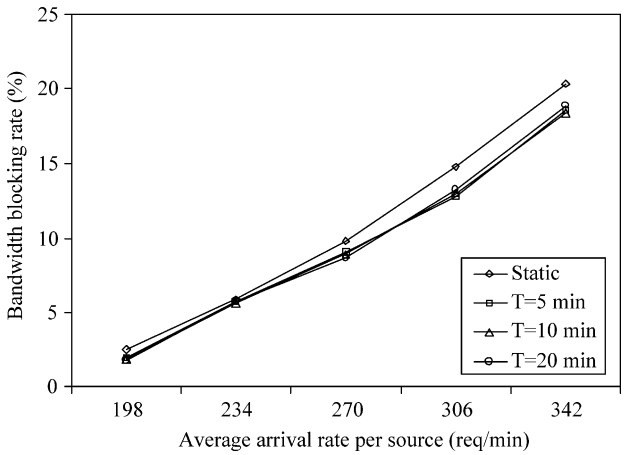

Fig. 7. Effect of the reallocation period T on the bandwidth blocking rate (unicast flows).

by running the resource allocation algorithm at the start of each experiment (using the known values of λ_i for each source), and then using the resulting $\{x_i(l,m)\}$ values for admission control. Notice that for unicast flows, where the traffic load matrix is relatively stable, the static and dynamic resource assignment have very similar performance. On the other hand, for multicast flows, where every new request is destined for a large number of receivers, the resource reallocation procedure is essential in order to keep up with the constantly changing traffic patterns. Another promising result in Fig. 8 is that the performance of our scheme is not affected by the length of the measurement window, as long as it is kept at reasonably small values. Therefore, the control overhead imposed by the resource reallocation procedure is very low.

Finally, we compare our proposed traffic engineering mechanism with traditional centralized shortest path routing. For unicast flows (Fig. 9) our scheme clearly outperforms shortest path routing. Its bandwidth blocking rate is around 10% lower at high traffic loads, while at the same time it can maintain the average link utilization at lower levels. This result is quite normal, since shortest path routing tends to overload the links that belong to the shortest paths, and further connections have to be established over longer paths, which consume a lot of the network resources. For multicast flows (Figs. 10 and 11) the performance of the two schemes is almost identical, since the construction of a shortest path tree is more straightforward and involves a large portion of the network. Notice, that the 'unicast' curve

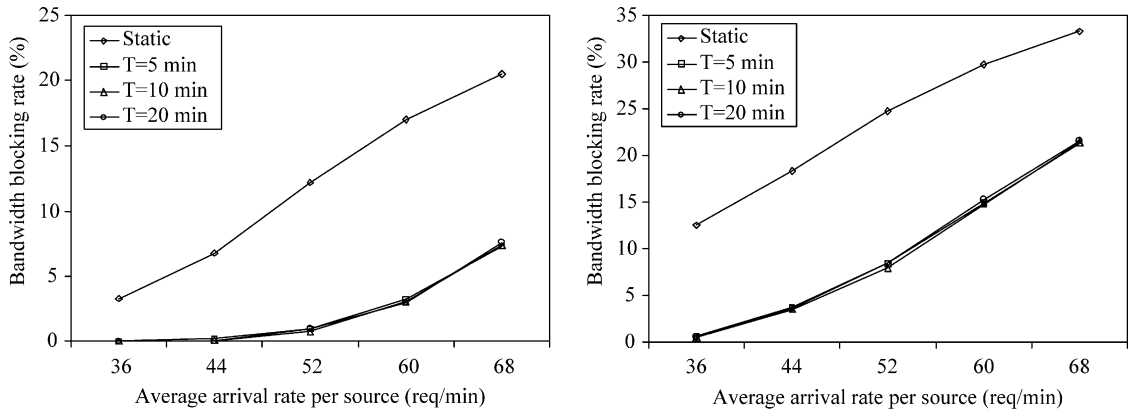

Fig. 8. Effect of the reallocation period T on the bandwidth blocking rate (multicast flows).

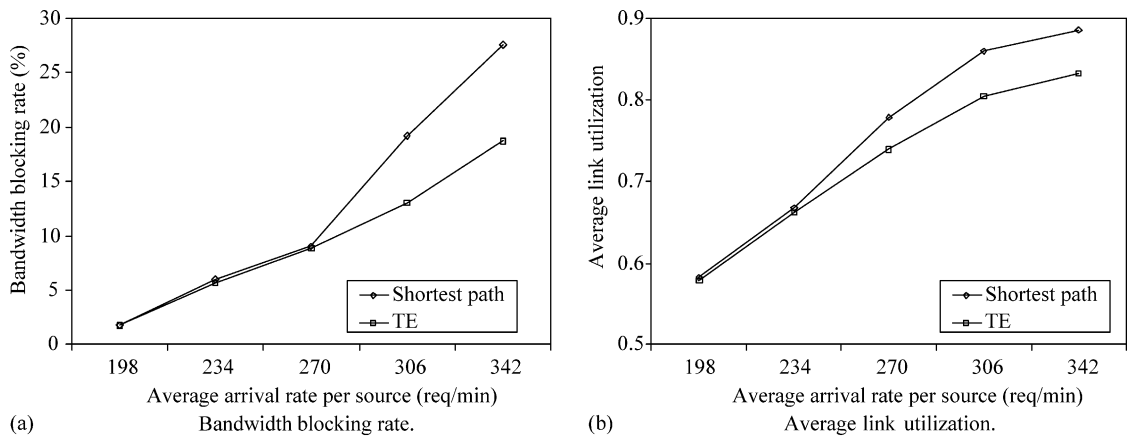

Fig. 9. Traffic engineering vs. shortest path routing (unicast flows).

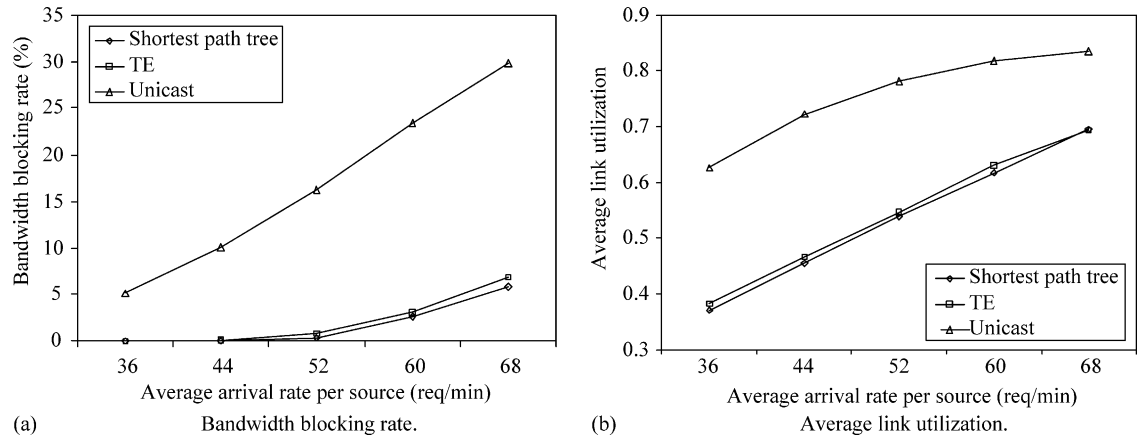

Fig. 10. Traffic engineering vs. shortest path routing (multicast flows, six destinations).

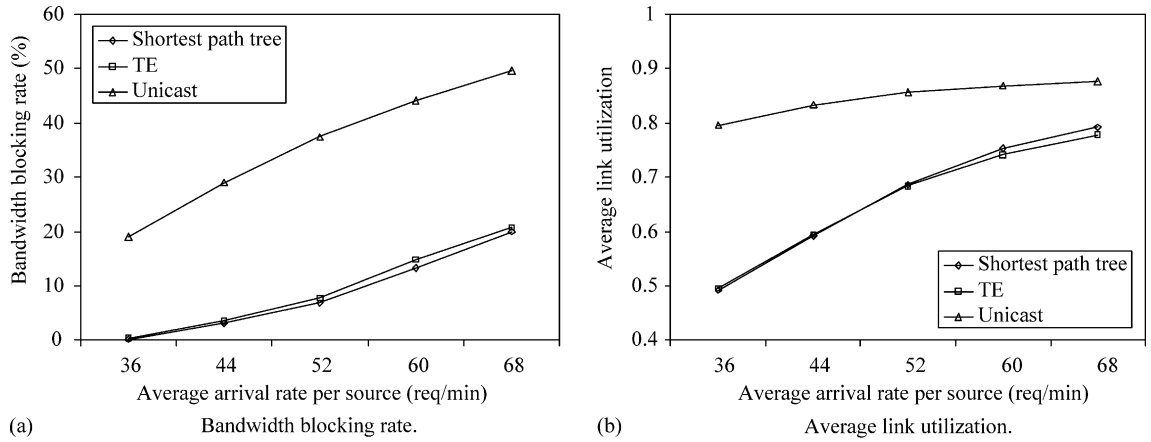

Fig. 11. Traffic engineering vs. shortest path routing (multicast flows, nine destinations).

corresponds to the case where each multicast request is served by a number of individual unicast connections. It is included in these figures for the purpose of comparison.

5. Conclusions

In this article we proposed a scalable architecture for providing end-to-end QoS guarantees in DiffServ-based networks. Our approach enhances the scalability of both the data and control planes, by pushing most of the functionality to the edge of the network. Furthermore, we introduced a traffic engineering mechanism, which periodically updates the allocation of the resources inside a domain, according to some measurements of the traffic load at the edge of the domain. We have shown that the proposed resource allocation algorithm can manage the network resources very efficiently, leading to lower bandwidth blocking rates compared to traditional shortest path routing.

We are currently working on the implementation of the architecture in a small testbed consisting of 19 Linux routers. Our goal is to test the applicability and performance of this architecture in an Internet-like environment. In addition, we will study the impact of QoS-sensitive flows on the performance of best-effort traffic, and investigate whether traffic engineering can improve the average delay experienced by best-effort packets inside a DiffServ domain.

Acknowledgements

This research is supported in part by the Areas of Excellence Scheme established under the University Grants Committee of the Hong Kong Special Administrative Region, China (Project No. AoE/E-01/99).

References

[1] S. Blake, D. Black, M. Carlson, E. Davis, Z. Wang, W. Weiss, An architecture for differentiated services, Internet RFC 2475.
[2] K. Nichols, S. Blake, F. Baker, D. Black, Definition of the differentiated services field (DS field) int the IPv4 and IPv6 headers, Internet RFC 2474.
[3] V. Jacobson, K. Nichols, K. Poduri, An expedited forwarding PHB, Internet RFC 2598.

[4] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, Assured forwarding PHB group, Internet RFC 2597.
[5] K. Nichols, V. Jacobson, L. Zhang, A two-bit differentiated services architecture for the Internet, Internet RFC 2638.
[6] B. Teitelbaum, S. Hares, L. Dunn, R. Neilson, V. Narayan, F. Reichmeyer, Internet2 QBone: building a testbed for differentiated services, IEEE Network (1999) 8–16.
[7] A. Terzis, L. Wang, J. Ogawa, L. Zhang, A two-tier resource management model for the Internet, in: Proceedings Global Telecommunications Conference (GLOBECOM), 1999, pp. 1779–1791.
[8] C. Chuah, L. Subramanian, R.H. Katz, A.D. Joseph, QoS provisioning using a clearing house architecture, in: Proceedings International Workshop on Quality of Service (IWQoS), 2000, pp. 115–124.
[9] E. Rosen, A. Viswanathan, R. Callon, Multiprotocol label switching architecture, Internet RFC 3031.
[10] T. Li, Y. Rekhter, A provider architecture for differentiated services and traffic engineering (PASTE), Internet RFC 2430.
[11] P. Trimintzios, I. Andrikopoulos, G. Pavlou, P. Flegas, D. Griffin, P. Georgatsos, D. Goderis, Y. T'Joens, L. Georgiadis, C. Jacquenet, R. Egan, A management and control architecture for providing IP differentiated services in MPLS-based networks, IEEE Commun. Mag. (2001) 80–88.
[12] Y. Rekhter, T. Li, A border gateway protocol 4 (BGP-4), Internet RFC 1771.
[13] J.Y. Yen, Finding the K shortest loopless paths in a network, Manage. Sci. 17 (11) (1971) 712–716.
[14] D. Bertsekas, R. Gallager, Data Networks, second ed., Prentice-Hall, Englewood Cliffs, NJ, 1992.
[15] Q. Ma, P. Steenkiste, On path selection for traffic with bandwidth guarantees, in: Proceedings International Conference on Network Protocols (ICNP), 1997, pp. 191–202.