Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computer Networks xxx (2004) xxx–xxx
COMPUTER NETWORKS
www.elsevier.com/locate/comnet

Extending differentiated services architecture for multicasting provisioning

Stavros Vrontis *, Efstathios Sykas

Electrical and Computer Engineering, National Technical University of Athens, 9 Heroon Polytechniou Str, 157 73 Zografou, Athens, Greece

Received 19 November 2003; received in revised form 24 September 2004; accepted 1 November 2004

Responsible Editor: I. Nikolaidis

Abstract

This article deals with the integration of multicasting in Differentiated Services (DiffServ) networks. We shortly introduce the main problems and the research activities in this area, in order to define the problem and give a clear picture of the current state-of-the-art. Then, we propose a solution, which provides the multicast functionality and conforms to the DiffServ architecture's basic principles. The main idea is to extend the typical DiffServ components to support multicast operations. We also present an algorithm for the calculation of the minimum-cost tree, which connects the source with the destinations, taking into account the available bandwidth per service class. Important implementation issues are analyzed and it is shown how the whole architecture can be deployed. Moreover, we evaluate the proposed mechanism, using a number of experiments and simulation tests. Finally, we compare our framework with other QoS provisioning schemes.

Keywords: Multicast; QoS; DiffServ; Active networks

1. Introduction

Multicast provides efficient packet delivery to a number of group members minimizing the total consumption of bandwidth [1]. The idea is, when transmitting the same packets to r receivers, to avoid creating r traffic flows and create just one flow instead. The flow will be replicated to the nodes that guide to different paths in order to reach the receivers. This way, bandwidth economy is achieved.

The integration of multicasting in DiffServ networks is not an easy task. The users require the

* Corresponding author. Tel.: +30 2107721511; fax: +30 2107722534.
E-mail address: stvront@telecom.ntua.gr (S. Vrontis).

1389-1286/$ - see front matter
doi:10.1016/j.comnet.2004.11.001

traffic packets to be delivered with a certain Quality of Service (QoS) level. The main problems arise because of the contrast in the basic principles of DiffServ and multicasting. The basic principle of the DiffServ logic is that the complexity and the state information involve only the edge routers in a DiffServ domain [2]. The flows that enter a DiffServ domain are shaped/policed in the edges, according to the predefined Service Level Agreements (SLAs), while the core nodes simply classify the packets according to their IP header (DS byte) and forward them with different priorities. No state information per flow is kept in the core nodes and all the complex functions are pushed to the edges (ingress and egress nodes). On the other hand, multicast requires all the multicast capable nodes to store information state per multicast group. The main problems of multicast and DiffServ integration are related to:

- *Scaling problems*, because of the per-group information storage in all multicast nodes, which comes in contrast with the statelessness of the core routers in DiffServ.
- *Receiver-driven QoS* for multicasting, instead of sender-driven QoS for the DiffServ architecture. Within DiffServ the sender starts the QoS provisioning procedure, in the sense that the sender (or the sender's gateway router) marks the packets with a specific Differentiated Services Code Point (DSCP) determining the QoS level. On the other hand, the procedure of QoS provisioning within multicasting is receiver-driven, since initially the receiver joins the multicast group requiring a specific QoS level.
- *Resource management*: Within multicasting, a packet can be replicated once or more in several nodes, creating this way a traffic distribution tree. Within multicasting in a DiffServ domain, the ingress router polices and shapes the incoming traffic according to the SLAs, without having complete knowledge of the multicast tree. Therefore, the ingress router cannot foresee the resources' consumption in all the network links. When a new receiver joins the multicast group, the current multicast tree is extended with a new branch, which connects the new receiver to it. Hence, resources are also consumed in the links of this branch, without having been reserved, affecting negatively the other flows, which have already reserved bandwidth. This problem is known as the NRS problem (Neglected Reservation Sub-tree) [3].
- *Heterogeneous QoS*: Within DiffServ, the sender transmits packets with a certain DSCP code for the DS byte, which determines the provided QoS level, without considering the receivers' capabilities. On the other hand, the receivers may require different QoS levels. Therefore, mechanisms should be foreseen for service differentiation to the tree branches.

2. State-of-the-art

Many researchers deal with the problem of the integration of multicasting with DiffServ. The current solutions can be classified into three main categories: (a) state-based, (b) edge-based and (c) encapsulation-based. Unfortunately, all solutions in these categories suffer from different disadvantages and only solve the problems partially.

The state-based solutions require all the routers (edge and core) to store state information per multicast group. The solutions of this category do not scale well and also violate the basic principle of DiffServ logic, which is the statelessness of the core routers. The IETF-RFC by Bless et al. [3] proposes such a solution. It focuses on the NRS problem, but the proposed solution for supporting heterogeneous QoS is oversimplified. Another solution of this category is included in the QUASIMODO approach [4]. QUASIMODO proposes extensions to the PIM-SM [5] multicast model, with the use of the GRIP (Gauge and Gate Reservation with Independent Probing model), in order to offer dynamic resource allocation. *QoS state* (QoS/No QoS) is also introduced for supporting heterogeneous QoS. The authors of [6] deal mostly with the NRS problem and receiver-driven QoS. They propose the DAM (DiffServ Aware Multicasting) solution. Their approach is based on weighted traffic conditioning, receiver-initiated marking and heterogeneous DSCP encapsulation. The

weighted traffic conditioning technique reserves bandwidth by overestimating the bandwidth consumption on the links, based on the traffic flows that enter and the number of copied flows that leave the domain. Even though the SLAs are never violated within this approach, the reserved bandwidth may never be consumed.

Li and Mohapatra have presented another state-based solution: the QMD (QoS-aware Multicasting in DiffServ Domains) multicast routing protocol [7]. Within their approach, the edge routers are supposed to know the exact network topology and the available network resources for each link. The ingress router calculates the multicast tree that connects the source with the receivers utilizing a QMD-DIJKSTRA proposed algorithm. The ingress nodes also need to maintain a Multicast Group Table (MGT), which contains records with multicast group information. The ingress node installs a multicast tree in the sense that specific core nodes are configured as *key node*s to maintain group information. The source sends the data in unicast packets to the first key node. The key node replicates the received packets and sends them to its children keys with simple unicast transmission. Each key node replicates and marks the incoming packets providing service differentiation to the tree branches. Our proposed solution inherits the QMD solution concerning the unicast transmissions. However, the QMD approach is incomplete, as far as issues like admission control, resource management etc are concerned, while it also does not deal with all the multicast and DiffServ problems. Even though the QMD approach moves complexity to the edges, it still demands state information maintenance in some core routers.

Within the edge-based solutions, the core routers are not multicast capable and the multicasting functions are limited to the edge routers. No trees are built and no packet replications are performed in the core network. These operations are only available in edge routers. Obviously the bandwidth usage is not the most efficient, since no multicast is actually performed inside the DiffServ domain. The most typical algorithm of this category is the EBM (Edge-Based Multicasting) [8]. In EBM the packets are tunneled between the edge nodes, removing entirely the concept of multicasting from the core network. The distribution tree is constructed utilizing the Edge Cluster Tree algorithm, which provides heterogeneous QoS routing.

The encapsulation-based solutions are based on the extension of the IP header of the multicast packets, in order to include the tree topology. The routers parse the IP header and decide whether to replicate the incoming multicast packet or not, determine the number of copies and select the proper DSCP value of the DS byte for each copy. This approach does not scale well because the IP header, and consequently the total length of the multicast packet, increases proportionally to the number of the receivers. In case there are many receivers, the extra bandwidth consumption from useless information grows significantly, reducing the algorithm's performance. Moreover, the routers' CPU utilization increases because the routers perform extra processing and complicated calculations. The main representatives of this category are the DSMCast (DiffServ MultiCast) [9] and the XCAST [10] protocols.

Taking into account the benefits and weaknesses of the above three categories, we ended to a solution, which we present to the following paragraphs.

3. The proposed solution

In this paragraph we present our approach to the integration problems of multicasting in DiffServ domains. The basic idea is to extend the functionality of the DiffServ components without violating the basic principles of the DiffServ architecture, in order to provide multicasting.

The proposed multicast data forwarding is based on a mechanism similar to the QMD (Fig. 1). The Bandwidth Broker (BB) is responsible for the construction of the multicast tree. The problem of the multicast tree construction can be described as: Given a source and r receivers with specific QoS level requirements, specify the nodes/links of the DiffServ domain that can connect the source with the destinations, without violating the available resources, consuming totally the minimum

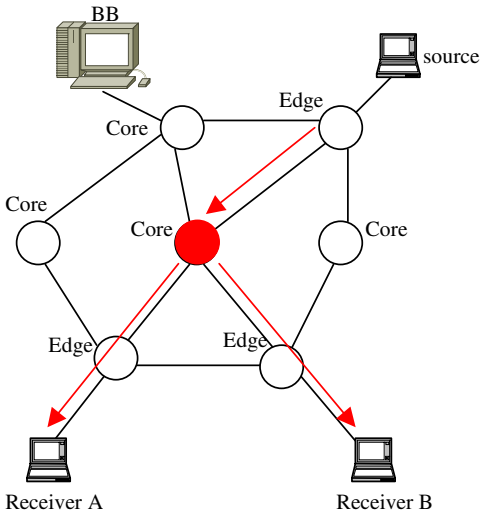

Fig. 1. Multicast equivalent with unicast transmissions. The source sends packets to the key-node (red node) with unicast transmission. The key-node replicates the received packets and sends each copy to receivers A and B with unicast transmissions. (For interpretation of the references in colour in this figure legend, the reader is referred to the web version of this article.)

network resources. The source is the root of the tree, while the receivers are the leaves. The idea is to provide the multicasting functionality using unicast transmissions between specific nodes of the multicast tree. These key-nodes are the nodes of the tree with more than one child-branch.

The basic components of a DiffServ domain are the core routers, the edge routers (ingress and egress), and usually a Bandwidth Broker, which performs admission control functions. In our approach we keep this architecture, extending the DiffServ components in order to perform multicast functionality. We shortly report the new features we added to these components:

- *Core Router*: A *multicaster* module is installed in all the core (and edge) routers of the domain. A multicaster receives unicast packets from its parent (source or multicaster) and forwards them to its children (multicaster(s) or receiver(s)). The multicaster sets the appropriate DSCP to each copy before it sends it to the corresponding child. Moreover, the multicaster may store the incoming packets for a time period (*buffering mode*), instead of immediately transmitting the copies to its children nodes (*active mode*). A multicaster module is launched in each key-node. The execution or the termination of a multicaster is triggered by the reception of a specific signaling message, which is sent by the source's neighbor edge router. More than one multicaster may be running within a router, if the router is a key-node for more than one multicast groups.
- *Edge Router*: The edge routers function as intermediate components between the users (source or destinations) and the BB. A user has to notify its neighbor edge router when he wishes to join or leave a multicast group utilizing the IGMP protocol [11]. Edge routers also receive transmission requests to multicast groups. All the requests are forwarded to the BB. The edge router, which is adjacent to the source, configures the multicast tree nodes (multicasters' launching/routing tables' configuration) by sending signalling messages to the tree's nodes. Multicaster modules may be launched in any edge router, if necessary. One is always launched in the edge router, which is adjacent to the source, even if this multicaster has only one child-multicaster. The source's multicast packets are captured by its neighbor edge router and forwarded to the local multicaster.
- *Bandwidth Broker*: The BB knows the exact network topology and is able to calculate the available bandwidth for each link in the DiffServ domain. For simple unicast transmissions, the BB is informed about the ingress and egress traffic by the edges and calculates the available bandwidth (based on the exact network topology, the links of the path that the traffic flow traverses and the initial available bandwidth for each link per service class). The BB also calculates the multicast trees and stores the relevant information. The BB uses this information to calculate the bandwidth consumption on the domain's links during multicast sessions. The multicast tree information is used to export information for the exact path that the multicast packets will follow and the multicaster nodes. The BB forwards this information as a

multicast tree object to the source's edge router, in order to install the multicast tree (i.e. to configure the routing tables of the tree nodes and launch the multicasters in the key-nodes).

3.1. Group membership

The receivers can request to join a multicast group by sending an "IGMP Join" message to their neighbor edge router. We propose the revision of the IGMP protocol for supporting service differentiation with modification of the "IGMP Join" message adding a new field for the required DSCP byte for the IGMP version 2.

Alternatively, for IGMP version 1 (Table 1), the "Unused" field of an "IGMP Join" message could be used for the DSCP byte.

Once an edge router receives an "IGMP Join" message, it encapsulates it in a signaling packet (e.g. UDP) and transmits it to the BB (Fig. 2). The BB keeps a table, which contains the members of each group with the required DSCPs (Table 2).

When a receiver leaves the multicast group, it informs the edge router, which forwards the (encapsulated) "Leave Group" message to the BB. The edge router also sends periodically "IGMP Membership Query" messages to its subnet. If a group member does not reply with an "IGMP Join" message, the edge router will consider that the member has left the multicast group and it will send a "Leave Group" message to the BB.

3.2. Resource management

The traffic that enters a domain may be unicast or multicast. In case of unicast traffic, the consumption of the bandwidth resources to the domain links may easily be estimated. The ingress node meters the unicast traffic and informs the BB about the consumed bandwidth, the service class and the destination of the packets flow. The BB keeps a topology table, which contains all the domain's links with the available bandwidth for each service class (Table 3). It calculates the path that the unicast traffic will follow to reach its destination and subtracts the con-

Table 1
IGMP version 1 packet format

00 01 02 03	04 05 06 07	08 09 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Version	Type	Unused	IGMP Checksum
Group Address			

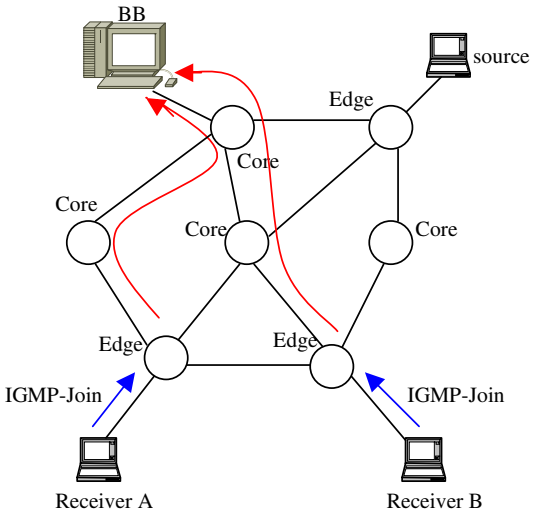

Fig. 2. Each time a user wishes to join a multicast group, he sends an IGMP Join message to the nearest edge router. The edge router forwards this message to the BB.

Table 2
Groups-Members table in BB

Group	Members
224.24.56.12	147.102.7.45-AF11, 147.102.8.89-EF, 147.102.5.23-EF
224.25.23.12	147.23.34.23-BE, 147.102.7.89-EF, 147.102.7.12-BE

Table 3
The BB keeps a "Links-Available Bandwidth" table for all the domain's links

Links	EF class (Mbps)	AF class (Mbps)	BE class (Mbps)
A → B	2	1	3
B → C	0.1	1	5
B → D	1	0.2	3
C → E	2	1	2

sumed bandwidth for the path's links for the specific service class.

In case of multicast traffic, the user who wishes to transmit to a multicast group sends a multicast transmission request message to its neighbor edge router, which then forwards the request to the BB. The BB retrieves the group members from the Groups-Members table (Table 2). Then, the BB finds the minimum-cost distribution tree utilizing an algorithm, which is based on the Dijkstra Shortest Path algorithm [12] for the computation of the shortest path between the source and the destination. We present an innovative heuristic algorithm for the Multicast Tree Calculation in the following paragraph.

Furthermore, the edge routers notify the BB for receiver events ("Join/Leave Group") or source events ("Start/Stop Transmission"). Hence, the BB is able to re-calculate the multicast tree and update the "Links-Available Bandwidth" table.

The proposed architecture solves the NRS problem, since the BB estimates the bandwidth consumption on all links and therefore bandwidth violations are avoided. Furthermore, our heuristic algorithm that is presented in the next paragraph provides a satisfactory solution to the tree optimization problem (Steiner tree problem) with bandwidth constraints.

3.3. The Multicast Tree Calculation Algorithm

The Multicast Tree Calculation Algorithm (MTCA) aims to produce the minimum-cost tree, which connects the source with the group's members. Initially, the BB creates the cost table from the "Links Available Bandwidth" table (Table 3); it assigns the NORMAL_COST value (e.g. 100) to a link, if the requested rate is less than the available link's bandwidth for the specific service class, otherwise it assigns the INFINITE_COST value (e.g. 100,000) to it. For example, if the source requests a 1 Mbps transmission rate, the BB will create the Table 4 for the case of Table 3.

The BB classifies the receivers into three classes for each service class: Expedited Forwarding (EF),

Table 4
The BB creates a cost table for the domain

Links	EF Class	AF Class	BE Class
A → B	100	100	100
B → C	100,000	100	100
B → D	100	100,000	100
C → E	100	100	100

Assured Forwarding (AF) and Best Effort (BE). Initially, the BB calculates the multicast tree for the EF receivers and each link of the EF tree is tagged with the EF DSCP. Then, it calculates the multicast tree for the AF receivers, merges it with the EF multicast tree, and the links of the AF tree are tagged with the AF DSCP. Finally, the multicast tree for the BE receivers is calculated and it is merged with the current tree. Please note, that the tags of the tree links determine the QoS level for the tree's branches and are given as input to the multicasters in order to mark the replicated packets properly. The algorithm's steps are analyzed below:

- The shortest path is calculated for each destination, utilizing the Dijkstra's algorithm. The cost table gives the link costs. The selected path will not contain links that do not have the required available bandwidth. If the shortest path contains a link with cost equal to the INFINITE_COST, the receiver will be removed to a lower service class (except from the BE receivers). For example, an EF receiver will be removed to the AF receivers, if there is no path to satisfy its bandwidth requirements for EF traffic.
- The cost table is updated: The MIN_COST value (e.g. 60) is assigned to the path's links for all classes. The updated cost table will cause the BB to prefer paths that contain links that have already been selected in previous steps. The selection of the MIN_COST value influences the produced multicast tree. Normally, the total number of the links of the multicast tree is decreased when the MIN_COST value is lower than a specific threshold, which depends on the network topology. However, the maximum length of a path is increased for MIN_COST values below this threshold.
- The selected path is merged with the current tree.

3.4. Tree installation

When all the receivers are parsed, a multicast tree object is constructed. Then, the BB encapsulates it in a signalling packet (e.g. "Tree Create"), sends it to the source's edge router, and updates the tables with the available bandwidth for the domain's links. Once the source's edge router receives the multicast tree information, it installs the multicast tree. Specifically, the tree installation refers to the multicasters' launching and the tree nodes' routing table update. Concerning the multicasters' launching, the source's edge router sends to the key nodes a message with the command: Start multicaster m at UDP port $port_{gi}$, with parent multicaster m_p and children multicasters (m_{c1}, $m_{c2}, \ldots, m_{cx}$) with the corresponding $DSCPs$ ($DSCP_1, DSCP_2, \ldots, DSCP_x$) for each child multicaster. On the other hand, the routing table of all the tree nodes should be conformed to the tree structure meaning that the multicast packets should follow the paths that were defined by the BB, which are not necessarily the default/shortest paths but definitely include the links with the required (bandwidth) resources. The routing tables' setup can be done by sending packets, which contain commands for routing table setup. We discuss the possible implementation methods in Section 4.

Once the tree installation finalizes, the edge router returns to the source the "Multicast Reply OK" message, in order to start the transmission or send a "Multicast Reply NOT OK" in case of failure (for example, if there are no group members). These messages are custom and can be easily implemented (e.g. encapsulated in a UDP packet).

3.5. Data forwarding

If the source receives an affirmative message ("Multicast Reply OK") from the edge router, it starts the multicast transmission. The edge node shapes/polices the incoming packets and forwards them to the local multicaster. The local multicaster sends one unicast flow to each child-multicaster with the proper DSCP. Similarly, each multicaster replicates the received packets, sets their DSCP byte and sends the packets to its children multicasters. Concerning the selection of the DSCP value, the multicaster assigns to each replicated flow (directed to a subset of receivers) the DSCP value that corresponds to the highest service level for the

receivers' requirements (e.g. replicated packets that travel towards an Expedited Forwarding (EF) and a Best Effort (BE) receiver will be marked with the EF DSCP). The required DSCP value for each unicast flow is calculated within the multicast tree calculation by the BB (see Section 3). This information is included in the multicast tree object that is sent to the source's neighbor edge router.

Considering the example of Fig. 3, receivers A and C require BE service, the receiver B requires EF service and the receiver D requires AF11 service. The BB has constructed the tree: B → {C, Receiver-D}, C → {Receiver-A, G} and G → {Receiver-B, Receiver-C}. This tree has the minimum possible nodes. The links BC, CG and GRecvB have enough available bandwidth for EF traffic, the links BE and ERecvD have enough available bandwidth for AF11 traffic and finally the links CF, FRecvA and GRecvC have enough available bandwidth for BE traffic. If a link does not have enough available bandwidth, the BB should select a different path (for example, if BC link is congested with EF traffic, the BB could try the link BE with Router E as multicaster node), which satisfies the bandwidth requirements with the minimum cost. The source sends the packets to the edge router B. The ingress router (B) (which is also a multicaster node) polices the incoming traffic and forwards it to router C (multicaster-C) and to Receiver-D. The flow towards router C is marked with the EF DSCP, while the flow towards Receiver-D is marked with the AF11 DSCP. Multicaster-C creates two unicast flows: one flow with BE DSCP goes to Receiver-A and a second flow with EF DSCP goes to router G. Similarly, the multicaster in router G (multicaster-G) sends one flow to Receiver-B with EF DSCP and one flow to Receiver-C with the BE DSCP.

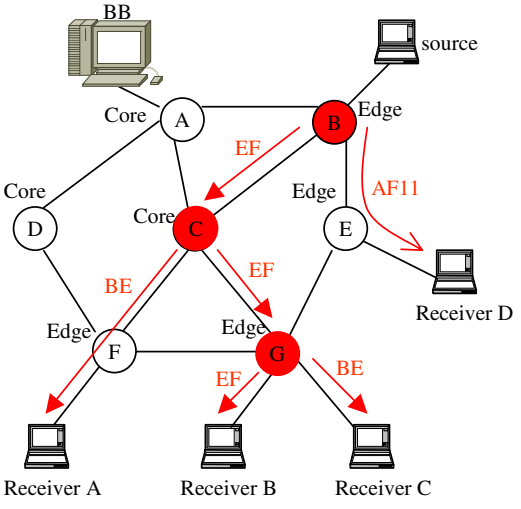

Fig. 3. QoS heterogeneity (multicaster modules run in the red nodes). (For interpretation of the references in colour in this figure legend, the reader is referred to the web version of this article.)

The QoS provisioning is receiver-driven since the receivers declare their QoS requirements before joining the group. Furthermore, QoS heterogeneity is achieved since the multicasters are able to re-mark packets with the proper DSCP.

3.6. Tree maintenance

The statelessness condition for the core network is satisfied, since no state information is maintained in the core nodes in the sense that multicasters are simply launched in the core network with the proper arguments by the BB, without maintaining this information (e.g. no timers are started, no callback functions are included waiting for "new arguments" events, no synchronization functions are included in order to update the state information etc). The necessary group's information is maintained exclusively by the BB. In case of multicast transmission termination, the source sends a "Terminate Transmission" message (e.g. a custom message encapsulated in a UDP packet) to its neighbor edge router. The edge router then informs the BB, in order to update the bandwidth tables and the multicast group's information. The BB identifies each multicast session by the group multicast IP address and the source IP address (G, S). Note that multicasters automatically die if no packets are received in a specified time period (e.g. 10 s).

When a receiver joins/leaves the multicast group, the BB will recalculate the multicast tree, in order to include/exclude him. Theoretically, the new tree could comprise of totally different multicasters. However, in case of a "Join Group" event, this is avoided because the BB parses the

new receiver after all the others (receivers with the same QoS requirements) during the multicast tree calculation procedure. Hence, the new path that guides to the new receiver will be merged with the current sub-tree. This way, the modifications of the multicast tree structure are minimized.

Once the BB has updated the multicast tree, it sends the new multicast tree object to the source's edge router, which re-installs the multicast tree "from scratch". A new multicaster may be launched in a key-node where an old one is already running, using, of course, a different UDP port for the reception of data packets. Note that these UDP ports are also assigned by the BB, which is also aware of the available UDP ports for all the domain's routers. For the feasibility of this operation, we propose the reservation of a UDP ports' area exclusively for the multicasters' operation.

Once the updated tree is installed, the source's edge router updates the children of the local multicaster (tree switch). This switch (from the current tree to the new one) is done instantly. The idea of re-installing the tree seems "greedy" concerning the number of the required signaling messages. However, it is simple to implement, the source and the current receivers are not influenced from "Join/Leave Group" events and packet losses are avoided.

After the tree switch, the old tree becomes dangling because no data packets are transmitted through it. However, a dangling multicaster automatically dies as it was mentioned earlier.

3.7. Inter-domain multicasting

Our proposal can be extended to support inter-domain multicasting. Let us consider the case that the source and the receivers are located in different DiffServ domains. One BB administers each domain with the functionality that was discussed in the previous paragraphs. Moreover, the BBs can communicate with each other.

Each BB knows the groups that the users of its domain have joined. When a source sends a message for multicast transmission request, the BBs communicate with each other, in order to discover which domains contain receivers for the specific multicast group. For the BB that administers the source's domain, the problem is defined as: Find the best multicast tree to connect the source with the $N-k$ local receivers and the l egress routers that guide to the networks of the k remote receivers ($l \leqslant k$). For a BB that belongs to an intermediate domain the problem can be defined as: Find the best multicast tree to connect the ingress node (which is boundary with the source's network) with the $N'-k'$ local receivers and the l' egress routers that guide to the networks of the k' remote receivers ($l' \leqslant k'$). The problem is degraded to n single-domain problems for the BBs that control the related domains. Each BB calculates a partial distribution tree, taking into account the available local resources and the local group members. Furthermore, the functionality of a boundary router's pair (between two neighbor domains) is similar to that of the source-edge router pair's, where the egress boundary node plays the role of the source, while the ingress boundary node plays the role of the source's neighbor edge router.

However, in the case of many interconnected domains, which are not serially connected, the problem is not simple and new issues should be considered. A BB has information only for its domain. Hence, the BB cannot take into account information from the other domains (for example the available bandwidth per link) for the selection of the egress routers that guide towards the remote destinations. Ideally, the BB should avoid "overloaded" domains. Furthermore, the BB is not aware of the overall network topology. Therefore, the BB cannot make efficient selection for the egress routers and the achieved bandwidth economy is not the optimal. The problem could be solved with a hierarchical approach. An overall BB, which communicates with all the BB, could provide to the BBs the required information.

4. Implementation issues

Concerning the realization of the architecture we presented in this article, we have already proceeded to the implementation of the BB. The implemented BB module acts as a server, which receives "Join/Leave Group" and "Start/Stop

Transmission'' requests by the edge routers for a multicast group, calculates the multicast group's tree, sends the multicast tree object to the source's edge router and keeps/updates the available bandwidth tables. The BB keeps information for more than one multicast group and communicates with the edge routers via TCP connections.

Within our proposed framework the required operations are balanced between the edge routers and the BB. Comparing the extended BB presented in this paper with a conventional one, the BB additionally undertakes the multicast tree calculations and keeps the necessary information for the multicast groups. On the other hand, the edge routers perform the multicast tree installation. These tasks' assignment is justified considering that the BB handles all the multicast groups in the domain and therefore we should prefer to assign jobs to the edge routers instead of overloading the BB. However, obviously in case of huge networks and many multicast groups with many receivers per group, the BB's performance will drop significantly. In such a case, more BBs could be used instead of one, with a central BB that redirects the tree calculation requests to the rest BBs, thus sharing this way the working load. The central BB will manage the tables and current copies of the tables could be passed to the other BBs. Within this approach, the main goal is to achieve synchronization, meaning that each BB must have the correct/updated information at any moment. The solution's complexity increases while the number of the BBs increases. The scalability issue and the limitations are analyzed using simulation tests in the following paragraph.

Regarding the multicast tree installation procedure (routing table configuration and multicasters' launching), three methods are considered: (a) SNMP-based method, (b) encapsulation-based method and (c) Active Networks (AN)-based method. In [13] we discuss these methods in detail and evaluate them in terms of bandwidth and time consumption. Within this paragraph, we briefly highlight the logic of each method.

Within the SNMP-based method, the source's edge router utilizes the SNMP protocol to configure one by one all the tree's routers. This method is simple and easy to implement but has low performance in terms of bandwidth and time consumption.

The encapsulation based method is similar to the encapsulated based one that refers the multicast data delivery in DiffServ domains (see Section 2). Within this method one signaling packet that carries the multicast tree object traverses the multicast tree (it is separated in key-nodes). Specifically, the IP header of the packet is extended with the multicast tree object. Each node parses the IP header of the packet, in order to retrieve the configuration command (if any) and to create/send one or more signaling packet(s) if the current node is a key-node. The IP header of the new signaling packets is updated (only the information for the downstream sub-tree remains while moving towards the leaves of the tree). The same logic is explained in detail for the XCAST protocol in [10]. This method has greater performance than the SNMP-based one, but it is more difficult to implement.

The AN-based method is based on the Active Networks technology [14–16]. AN is a framework where network elements are programmable. Programs can be injected into the network and executed by the network elements, in order to achieve higher flexibility and to present new capabilities. Active Networks can work in two different ways: users can preprogram selected network routers with customized code before running an application, or they can program data packets, which then transport code to nodes along the way to their destination. Ideally, all the routers of the domain should be active nodes. An active node may be a Linux-based router that contains the Active Engine (AE) module or a commercial router (e.g. Cisco), controlled by a Linux machine that contains the AE. The AE module is fed with ''active packets'' which contain Java source code and executes this code. Once an active packet traverses an active node, it is redirected to the AE module. An active packet is a UDP packet, which is identified by an active node with the destination UDP port. Within the AN-based method, an active packet is sent from the source's edge router downstream. Each node captures the active packet and executes its source code. The source code may cause the execution of a multicaster, the configuration of

the local routing table and the transmission of one or more active packets downstream. This method is similar to the encapsulation-based one concerning the signaling packets' transmissions. The AN-based method has better performance than the SNMP-based one, but worst performance than the encapsulation-based one, due to the bigger packet size (the active packet includes the active header and source code).

Please note that the signaling messages (for any method) should be marked with the EF DSCP. Moreover, the BB should take into account the small bandwidth consumption for the EF class to the tree's links during the configuration procedure, because of the signaling messages. We intend to implement the tree installation mechanism using the AN-based method. Specifically, we will utilize the DINA active platform, which is still under construction. DINA extends the ABLE active platform [17–19] with new capabilities. The DINA platform comprises of a set of brokers that perform management operations and provide the programmer with a powerful API for the brokers' utilization. Hence, the configuration (e.g. routing table configuration) of an active node is easy, since relevant DINA methods are available and the programmer may use them, in order to create the active code for the tree installation. An active application for the configuration of a single path between two nodes is presented in [20]. The implementation of DINA is a task currently undertaken by the CONTEXT IST project [21].

5. Simulation analysis and evaluation

We simulated the proposed architecture using the network simulator ns-2 [22]. We performed several experiments, in order to validate the performance of the whole architecture. Concerning the network topology, we mostly utilized network instances from the SteinLib library [23]. SteinLib is a collection of Steiner Tree problems' instances that contains their optimal solution or the best solution found so far. This information was used in order to evaluate the MTCA's performance. Please note that the experiments described in the next paragraphs refer to the five networks selected from the SP test-set of the SteinLib collection and shown in Table 5, below, if not specified otherwise.

Moreover, we created custom network graphs using the Waxman model [24]. The Waxman model randomly distributes nodes over a rectangular coordinate grid. The Euclidean metric is then used to determine the distance between each pair of nodes. Two nodes are connected with a probability $P(u,v)$ that depends on their distance:

$$P(u,v) = \beta \cdot e^{-d(u,v)/(L \cdot a)}, \quad (1)$$

where $d(u,v)$ is the distance from the node u to v, L is the maximum distance between two nodes and α and β are parameters in the range $(0,1)$.

A Microsoft Windows PC with CPU speed 1.4 GHz (Centrino technology) and 1024 MB RAM memory was used for all the performed tests.

5.1. MTCA performance analysis

Initially, we performed tests in order to study the influence of the MIN_COST value on the tree optimization problem. We performed the tests for several network topologies and receivers. We define as optimal $\text{MIN\_COST}_{\text{opt}}$ values, the ones that give multicast trees with the minimum number of links. Our experiments showed that generally there is a threshold value T for the $\text{MIN\_COST}_{\text{opt}}$ values, which influences the path selection. The

Table 5
Five networks selected from the SP test-set of the SteinLib library

Name	Graph's nodes	Graph's edges	Receivers	Optimal no. of links
oddwheel3	7	9	4	5
antiwheel5	10	15	5	7
w13c29	783	2262	406	507
w23c23	1081	3174	552	692
w3c571	3997	10,278	2284	2854

tree's total cost is stable for MIN_COST values greater than T. For MIN_COST < T, the total cost is improved and is maintained the same for all values of MIN_COST between $\text{MIN\_COST}_{\text{opt}}$ and T. However, while the $\text{MIN\_COST}_{\text{opt}}$ values provide trees with less total links, the length of the longest path from the source to any destination increases. Even though in some experiments the algorithm calculated the same multicast tree for all MIN_COST values, we observed the threshold's existence in most of the cases. The experiments showed that the threshold's value depends on the network topology. Generally, we came to the following conclusion for the T values:

$$T < \frac{\text{NORMAL\_COST}}{2}. \tag{2}$$

The algorithm has better performance in dense networks. Furthermore, the threshold T depends on the order, in which the algorithm parses the receivers. We changed the receivers' order and got different threshold values. The threshold also depends on the receivers' position in the network. Fig. 4 shows the results of two experiments, which were performed using different network topologies. Both networks comprised 40 nodes, randomly connected (Waxman grid). For each network we performed the experiment for 4, 8 and 16 receivers, modifying the value of MIN_COST. For each case, the Tree Optimization Success Ratio (TOSR) represents the percentage of the theoretically minimum total number of the multicast tree's links, ML, to the total number of the multicast tree's links, L:

$$\text{TOSR} = \frac{ML}{L} \cdot 100\%. \tag{3}$$

Furthermore, we applied MTCA to the SteinLib instances for two cases of MIN_COST: (a) MIN_COST = NORMAL_COST/2 and (b) MIN_COST = NORMAL_COST/5. MTCA was applied to numerous subsets of the receivers. Initially, MTCA experiments included only the first receiver from the SteinLib network. Then, we increased the number of the receivers, by including the next receiver and repeated the experiment. The procedure was repeated and ended when all of the receivers were included. The number of links for the produced multicast trees appears in Fig. 5 (logarithmic scale for both axes).

The TOSR values for the SteinLib networks (with the maximum number of receivers) appear in Table 6. We see that MTCA has excellent performance for small networks, while the performance decreases for larger networks.

The experiments with random Waxman grids confirmed that our algorithm provides excellent performance to the tree optimization problem, since we had minimum TOSR values close to 75%.

5.2. MTCA time analysis

MTCA utilizes the Dijkstra's Shortest Path algorithm for the calculation of the shortest path between the source and each receiver. The running time of the Dijkstra-SP algorithm (the version that uses a Fibonacci heap for the storage of the vertices' labels) is $O(E + V \log V)$, where E is the number of edges and V is the number of vertices.

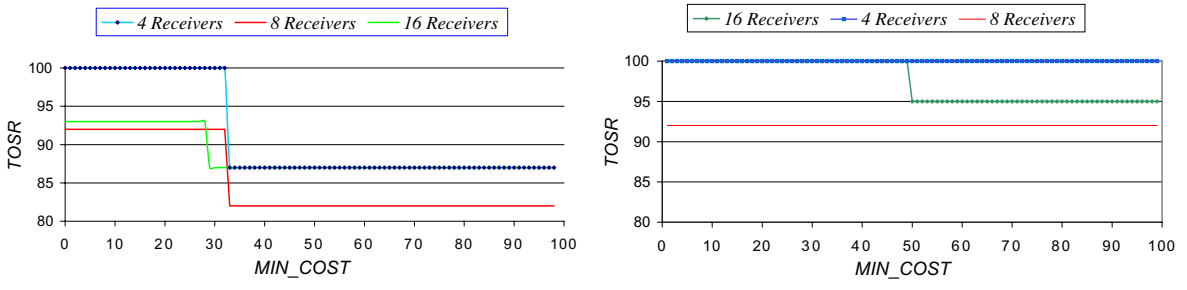

Fig. 4. Selecting values for the MIN_COST below 33, the MTCA has the maximum performance (left), while for another network (right) we should select values for the MIN_COST below 50 (NORMAL_COST = 100).

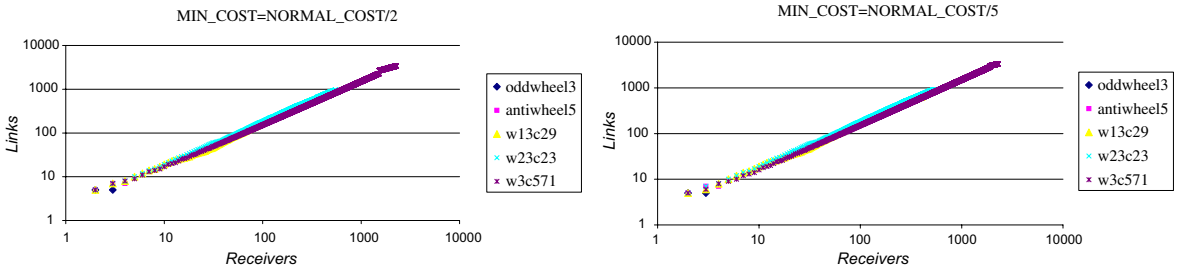

Fig. 5. The number of links of the produced multicast trees for SteinLib networks (left: MIN_COST = NORMAL_COST/2, right: MIN_COST = NORMAL_COST/5).

Table 6
TOSR values for the SteinLib networks

Name	TOSR (MIN_COST = NORMAL_COST/2) (%)	TOSR (MIN_COST = NORMAL_COST/5) (%)
oddwheel3	100	100
antiwheel5	100	100
w13c29	75	78
w23c23	73	75
w3c571	83	85

Moreover, the cost table is sorted alphabetically. Since the (binary) search for a link requires $O(\log E)$, the update of the cost table requires, in the worst case, $O(D \cdot \log E)$ for each receiver (where D is the network's diameter). The merging of a path with the current tree requires $O(D)$ (worst case). Please note that this time decreases, as the network graph becomes denser. Summarizing, it is easy to see that the required time for MTCA, in a graph with diameter D, V vertices, E edges and r receivers is

$$O(r \cdot (E + V \log V + D \log E)). \quad (4)$$

We performed tests for several network graphs (Waxman grids and SteinLib networks), in order to measure the required time for MTCA.The tests confirmed that the calculation time increases, while the number of receivers increases. Moreover, the required time increases, while the network's nodes/edges increase. The calculation time for the SteinLib networks (logarithmic scale for both axes) appears in Fig. 6.

We conclude that the required time for MTCA is less than one second in normal cases (regarding the number of group members), a few seconds in case of larger networks, while one minute is required in extreme (huge) cases. We propose the periodical calculation of the multicast tree, instead of calculating it "on demand" each time a "Join/Leave Group" event occurs. The period should be selected considering the usual number of receivers and the total nodes of the network. Observing Fig. 6, we conclude that a period of 10 s is satisfactory for most cases, while a period of 1 s should be selected in case of small multicast groups. The network's administrator should proceed to statistical studies concerning the number of receivers, in order to adjust the period for well-known multicast groups (e.g. popular seminars using videoconference applications).

Please note, finally, that when a user requests to join a multicast group, he will be able to join the group after a time period, which is equal to the tree calculation time plus the required time for the tree installation procedure.

5.3. Core network processing load analysis

We also performed tests in order to study the number of running multicasters during a multicast

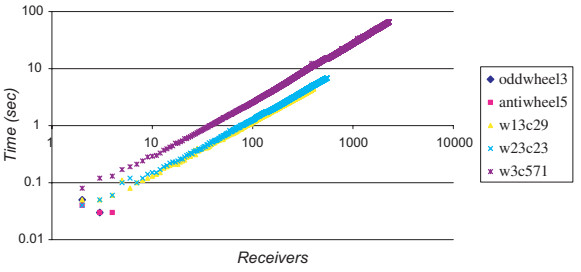

Fig. 6. MTCA's required time for SteinLib networks.

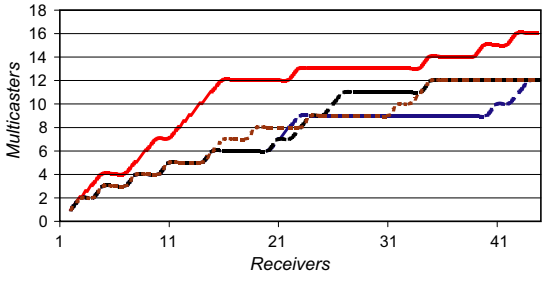

Fig. 7. The running multicasters in four scenarios: "Join Group" events occur with randomly selected receivers.

session. The experiments included "Join Group" events every 0.1 s and counting of the running multicasters. We observed that the number of multicasters increased proportionally to the number of the receivers. The results from four tests in a Waxman network grid ($V = 100$, $E = 85$) appear in Fig. 7.

We also present the number of multicasters for the SteinLib networks in Fig. 8.

5.4. Tree installation analysis

The performance of the tree installation procedure, in terms of time and bandwidth consumption, depends on the selected method (see Section 4). In [13], we prove that the mean required time $\overline{T^{\text{enc}}}$ for the tree installation and the bandwidth consumption are minimized when the encapsulation-based method is used.

Specifically, the mean required time $\overline{T^{\text{enc}}}$ is given by

$$\overline{T^{\text{enc}}} = (h+1)\cdot \overline{t_c^{\text{enc}}} + \frac{H_{\text{IP}}\cdot h}{R} + \frac{\overline{D^{\text{enc}}}\cdot C}{R} + \overline{t_{a_h}}, \qquad (5)$$

where h is the height of the multicast tree, $\overline{t_c^{\text{enc}}}$ is the mean required time for calculations at a tree node, H_{IP} is the length of the packet header ($H_{\text{IP}} = 46$), R is the transmission rate (e.g. 10 Mbps), D^{enc} is the length of the total multicast tree object, $\overline{t_{a_h}}$ is the mean required time for the configuration of the leaf node (h-node), and C is a factor that depends on the tree structure ($C < h$).

We simulated the encapsulation-based method considering that $R = 10$ Mbps for all links and that $D^{\text{enc}} = r \cdot 100$, where r is the number of the receivers. The tests showed that the $\overline{T^{\text{enc}}}$ is in the order of tenths of the second in case of small and medium network graphs (see SteinLib networks), while $\overline{T^{\text{enc}}}$ is a few seconds (less

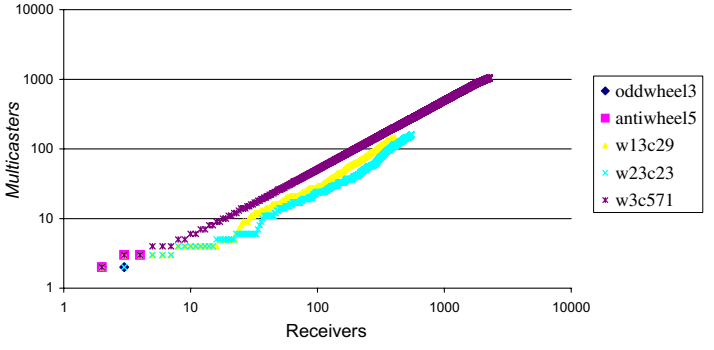

Fig. 8. The number of multicasters for SteinLib networks (MIN_COST = NORMAL_COST/2).

than 4 s for the case of the w3c571 SteinLib network).

In [13], we also prove that the total number of bytes that are transferred to the network (mean value) within the encapsulation-based method is given by

$$\overline{B^{\text{enc}}} = (N-1)\cdot 46 + \text{IPL}\cdot\left(\overline{D^{\text{SNMP}}} + 4\right), \tag{6}$$

where N is the number of tree nodes, $\overline{D^{\text{SNMP}}}$ is the mean length of the data part of an SNMP configuration message ($D^{\text{SNMP}} \approx 100$) and IPL is the internal path length of the tree. According to basic tree theory, for random trees:

$$\text{IPL} = \text{O}(N\cdot\log N). \tag{7}$$

Please note that the AN-based method has a slightly worse performance than the encapsulation-based one, concerning both variables (time and bandwidth), while both methods are significantly better than the SNMP-based one.

5.5. Join latency analysis

The join latency t_{join} refers to the total time required for a new group member to start receiving multicast data packets, counting from the moment that the user requests to join the multicast group through the IGMP protocol. We will consider two cases that correspond to different scenarios of implementation.

In the first case scenario, the BB updates the multicast tree once it receives the encapsulated "IGMP Join" message from the source's edge router. The t_{join} value is given by

$$t_{\text{join}} = t_{\text{ER}\to\text{BB}} + t_{\text{BB}} + t_{\text{BB}\to\text{ER}} + t_{\text{install}}, \tag{8}$$

where $t_{\text{ER}\to\text{BB}}$ is the required time for the transmission of the encapsulated "IGMP Join" message from the source's edge router to the BB, t_{BB} is the required time for tree calculation by the BB (MTCA), $t_{\text{BB}\to\text{ER}}$ is the required time for the transmission of the multicast tree object to the source's edge router, and finally t_{install} is the required time for the installation of the multicast tree.

Simulation tests showed that the $t_{\text{ER}\to\text{BB}}$ and $t_{\text{BB}\to\text{ER}}$ values are significantly lower than the t_{BB} value. Moreover, simulation showed that the $t_{\text{BB}\to\text{ER}}$ increases while the number of the receivers increases, but the t_{BB} increases faster and therefore we could always consider that

$$t_{\text{join}} \approx t_{\text{BB}} + t_{\text{install}}. \tag{9}$$

The study for the t_{BB} values appears in paragraph 5.2, while the study for the t_{install} value appears in paragraph 5.4. We conclude that generally $t_{\text{BB}} \gg t_{\text{install}}$ and therefore we can consider that the join latency can be simplified:

$$t_{\text{join}} \approx t_{\text{BB}}. \tag{10}$$

Please note that if more than one BBs are used for the calculation of the multicast tree, the required time for the message exchange should be added to (8).

The second implementation scenario is based on periodical updates of the multicast tree by the BB every T seconds. In this case, the join latency is a random variable with mean value $\overline{t_{\text{join}}}$:

$$\overline{t_{\text{join}}} = \frac{T}{2} + t_{\text{BB}} + t_{\text{install}} \approx \frac{T}{2} + t_{\text{BB}}. \tag{11}$$

6. Comparison with other QoS provisioning schemes

The proposed framework deals with all the integration problems between multicast and DiffServ, while all the others focus on one or a few of them. Moreover, it is flexible and extendable to solve other integration problems related with multicast (e.g. reliability issues, users' mobility and multicast, etc.). For example, the framework can be extended with mechanisms for reliable multicast provisioning considering that the multicasters are able to keep copies of the received unicast data packets and retransmit them in case of packet losses. Comparing it with the edge-based solutions, it is obvious, that our framework is more efficient concerning the bandwidth consumption. Within the edge-based solutions multicast functions are limited to the edge routers. Hence, in general the constructed multicast trees are not optimal by default. On the other hand, our solution is more complicated and requires the exchange of signaling messages. However, the bandwidth overhead due to the signaling messages is significantly less than

the total bandwidth consumption due to the inefficient tree structure of the edge-based solutions. Note that the bandwidth profit decreases when "Join/Leave Group" events occur with high frequency because of the required signaling messages. The edge-based solutions have better performance only for the special case that the optimal tree branches only in the edge routers and therefore it is identical to the tree that is formed within the edge-based method (for example: all the receivers are located in the same sub-network).

Within the encapsulation-based solutions (e.g. XCAST), the data packets include an extra header for routing information and therefore the bandwidth performance is low. Moreover, the header's size (and of course the total packet's size) increases proportionally to the number of the group members. Hence, these solutions are not scalable and cannot be applied to multicast trees with many receivers. Actually, the specifications of these solutions mention this problem and declare that they can only be applied for multicast packet delivery to a few receivers. Note that within our solution the configuration information is not included in the data packets but in the signaling packets, which are sent transparently once a receiver joins/leaves the multicast group. Hence, within the encapsulation-based solution less bandwidth is consumed than within our scheme only during the small period (comparing it with the total multicast session period) of the tree installation procedure.

Comparing our scheme with the Bless' solution and the QUASIMODO, we can see that the trees are not optimal due to the creation method: when a new receiver joins the multicast group, a branch is added to the nearest node of the current tree that connects the receiver to it. After a number of join group events, the constructed tree will deviate from the optimal, while within our proposal the tree is recalculated. Therefore, our method has better performance concerning bandwidth consumption, even though less bandwidth consumption due to signaling messages is achieved within these schemes. Moreover, denial of service for a new group member may occur even if there is a path between the source and the receiver with enough available bandwidth resources for the required service class. Specifically, within these schemes, a branch that connects the new receiver to the nearest tree node is temporarily added and it is examined if there are available resources for the specific path. If not, the receiver is not served. Within our approach, the receiver always joins the multicast group with the desired service level if there is at least one path between the source and the receiver with available bandwidth resources for the desired service class. Moreover, if there is no path with the available resources for the desired service level, the user is served with lower QoS level instead of not being served at all. Note that once a "Join/Leave Group" event occurs, the multicast tree is recalculated "from scratch". Thus if there are available resources for the desired service class at that moment, the user (that is currently served with lower QoS level than the desired one) will be served with the desired QoS level.

Even though our framework extends QMD, however we do not inherit all of its features. The main difference of our proposal with QMD is the update of the multicast tree during the multicast session. Within the QMD-DIJKSTRA algorithm, a new branch is added to the current multicast tree, when a new user joins the multicast tree, while within MTCA the tree is recalculated to include the new group member. Even though, the new branch is efficiently selected within QMD-DIJKSTRA, the cost of the multicast tree is generally more than the one that is produced within MTCA. Of course, the signaling requirements are more within our framework, however the multicast tree is cheaper in terms of bandwidth consumption and therefore the main goal of bandwidth economy is achieved.

Finally, we consider that our solution's main disadvantage, comparing it with the other schemes, is the join latency. A new group member has to wait until the updated multicast tree is reinstalled. However, we think that this is not critical and the waiting time is too small to be annoying for the new receiver.

7. Conclusions and future work

In this paper we presented an architecture that enables multicasting in DiffServ domains. We in-

serted multicast operations into basic DiffServ components without violating the principles of the DiffServ. The DiffServ logic requires statelessness for the core network, while the edge routers and the Bandwidth Broker perform more complicated actions. Therefore, we moved the complicated multicast operations to the BB and the edge routers. However, we injected multicaster modules in core routers. Of course, this does not violate the statelessness rule for the core network, since the multicasters are independent of the DiffServ functionality of the core routers and, furthermore, they do not store information for the multicast session. The BB keeps this information. We achieved receiver-driven and heterogeneous QoS inside DiffServ, utilizing the multicasters under the BB's control. We also solved the NRS problem by extending the BB capabilities, in order to be able to estimate the bandwidth consumption in all of the domain's links. The BB is the component that calculates the multicast trees and therefore it can estimate the available bandwidth in the domain's links.

Moreover, we introduced a heuristic algorithm for the Multicast Tree Calculation. MTCA aims to solve the Steiner tree problem with the links' available bandwidth constraints. According to the simulation experiments, MTCA's performance is very high (TOSR > 80%). The main disadvantage of MTCA is the required time period, which is analogous to the receivers' number. However, we believe that in the near future this will not be a problem, as the computers' CPU speed increases rapidly. Furthermore, we analyzed the critical Group Membership, Resource Management, Data Forwarding, Tree Installation, Tree Maintenance and Inter-domain Multicasting issues. We also simulated the proposed architecture and we concluded about crucial MTCA parameters and the behavior/performance of the architecture's components. Finally, we compared our scheme with other QoS provisioning ones. The important advantage of our framework is the bandwidth performance, which is actually the multicast's main goal.

Concerning MTCA, we intend to proceed further with more advanced theoretical studies utilizing Graph theory and compare it with relevant algorithms (e.g. QMD-DIJKSTRA). We have already implemented the BB and intend to fully implement the proposed platform in order to evaluate it in a real network environment. Initially, we need to investigate the tree installation issue in order to end up with the most efficient method. Moreover, we intend to enhance the reliability of our multicasting mechanism, by making multicasters able to keep copies of the received packets and re-transmit them in case of packet losses. Finally, we will study the case where the receivers (and the source) are mobile and investigate possible extensions to our framework for seamless handovers during multicast sessions.

References

[1] K.C. Almeroth, The evolution of multicast, IEEE Network 14 (1) (2000) 10–21.
[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, An architecture for Differentiated Services, IETF RFC2475, December 1998.
[3] R. Bless, K. Wehrle, IP multicast in Differentiated Services (DS) networks, RFC 3754, April 2004.
[4] G. Bianchi, N. Blefari-Melazzi, G. Bonafede, E. Tintinelli, QUASIMODO: QUAlity of ServIce-aware Multicasting Over Diffserv and Overlay networks, IEEE Network 17 (1) (2003) 38–45.
[5] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, L. Wei, Protocol Independent Multicast—Sparse Mode (PIM-SM): Protocol Specification, IETF RFC 2362, June 1998.
[6] B.B. Yang, P. Mohapatra, Multicasting in Differentiated Service domains, in: Proceedings of IEEE GLOBECOM 2002.
[7] Z. Li, P. Mohapatra, QoS-aware multicasting in DiffServ domains, in: Proceedings of Global Internet Symposium 2002.
[8] A. Striegel, A. Bouabdallah, H. Bettahar, G. Manimaran, EBM: Edge-based multicasting in DiffServ networks, in: Proceedings of Network Group Communications (NGC), Munich, Germany, September 2003.
[9] A. Striegel, G. Manimaran, DSMCast: A scalable approach for DiffServ Multicasting, Computer Networks 44 (6) (2004) 713–735.
[10] R. Boivie, N. Feldman, Y. Imai, W. Livens, D. Ooms, O. Paridaens, Explicit multicast (Xcast) basic specification, IETF Internet Draft <draft-ooms-xcast-basic-spec-05.txt>, August 2003, Work in Progress.
[11] W. Fenner, IGMP Internet Group Management Protocol, Version 2, RFC2236, November 1997.
[12] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA, 1997.

[13] S. Vrontis, S. Xynogalas, E. Sykas, Efficient mechanisms for multicast tree setup in differentiated services networks, Computer Communications, submitted for publication.
[14] D.L. Tennenhouse, D.J. Wetherall, Towards an active network architecture, Computer Communication Review 26 (2) (1996).
[15] L.T. David, J.M. Smith, W.D. Sincoskie, J.W. David, A survey of active network research, IEEE Communications Magazine 35 (1) (1997) 80–86.
[16] K.L. Calvert, S. Bhattacharjee, E.W. Zegura, J. Sterbenz, Directions in active networks, IEEE Communications Magazine 36 (10) (1998) 72–78.
[17] ABLE: the Active Bell Labs Engine. Available from: <http://www.cs.bell-labs.com/who/ABLE/>.
[18] ABLE: The Active Bell-Labs Engine for Network Management, demonstration at OpenArch'99, New York, NY, March 1999.
[19] D. Raz, Y. Shavitt, An active network approach for efficient network management, IWAN'99, Berlin, Germany, July 1999.
[20] S. Vrontis, I. Sygkoyna, M. Chantzara, E. Sykas, Enabling distributed QoS management utilizing active network technology, in: Network Control and Engineering for QoS, Security and Mobility II, Kluwer Academic, Dordrecht, 2003, pp. 139–151.
[21] The Context IST Project: Active Creation, Delivery and Management of Efficient Context Aware Services, IST-2001-38142-CONTEXT. Available from: <http://context.upc.es>.
[22] The Network Simulator ns-2. Available from: <http://www.isi.edu/nsnam/ns>.
[23] SteinLib Testdata Library. Available from: <http://elib.zib.de/steinlib>.
[24] Bernard M. Waxman, Routing of multipoint connections, IEEE Journal on Selected Areas in Communications 6 (9) (1988) 1617–1622.

Vrontis P. Stavros was born in Athens, Greece, in 1975. He received his Diploma in Electrical and Computer Engineering from the National Technical University of Athens (NTUA) in 1998. Since 2000 he has been working with the NTUA Telecommunications Laboratory as a research associate, and is currently pursuing the Ph.D. degree in the area of computer networks. He has been actively involved in several European research projects in the field of computer networks. His research work lies in the areas of quality of service, multicasting, active networks, mobile networks, network optimization and distributed architectures. He is a member of the IEEE and the Technical Chamber of Greece.

Efstathios D. Sykas was born in Athens, Greece, in 1956. He received the Dipl.-Ing. and Dr.-Ing. degrees in Electrical Engineering both from the National Technical University of Athens, Greece, in 1979 and 1984, respectively. From 1979 to 1984 he was a Teaching Assistant, during 1986–1988 he was a Lecturer, then an Assistant Professor, 1988–1992, Associate Professor, 1992–1996 and now Professor in the Division of Communications, Electronics and Information Engineering, Department of Electrical Engineering, National Technical University of Athens (NTUA). He has served as Director of the Computer Science Division 1997–2000 and since 1999 is a member of the board of Governors of ICCS. He is a reviewer for several international journals and a member of IEEE Standard 802 Committee. He is a member of IEEE, ACM and the Technical Chamber of Greece.