Quality of Service Multipath Multicast Protocol*

Pierpaolo Baccichet
Computer Science Department
University of Milan
via Comelico 39, I–20135
Milano, Italy
p.bacciche@tin.it

Elena Pagani
Computer Science Department
University of Milan
via Comelico 39, I–20135
Milano, Italy
pagani@dsi.unimi.it

Gian Paolo Rossi
Computer Science Department
University of Milan
via Comelico 39, I–20135
Milano, Italy
rossi@dsi.unimi.it

ABSTRACT

This paper reports the preliminary work on QoSM^{2}P [1], a multicast protocol for the Internet that supports QoS–sensitive routing. QoSM^{2}P has been designed to achieve a better usage of network resources.

The primary trouble encountered in QoS multicast is the "false rejection" of join requests generated by the clients, when the network load is particularly high. The works described in [2], [3] and [4] may fail in serving the user properly, because they consider only one path returned from the underlying unicast protocol, without checking the existence of other alternative paths. QoSM^{2}P introduces an extended mechanism for finding feasible paths, that checks several alternatives within the same join request procedure.

The QoSM^{2}P protocol also provides the possibility to reorganize the QoS requirements, without any disconnection from the group. This feature is useful in many applications like layered video, in which a user may change the desired quality of service at any time.

Categories and Subject Descriptors

C.2 [**Computer Systems Organization**]: Computer Communication Networks—*network protocols, protocol architecture*; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

General Terms

Multicast algorithms, Performance analysis

Keywords

multicast, multipath, reservation, protocol architecture, quality of service

*This work was supported by the MURST under contract no. MM09265173 "Techniques for end–to–end Quality–of–Service control in multi–domain IP networks"

NGC '02 Boston, Massachussetts USA

1. INTRODUCTION

Multicasting can be defined as the distribution of the same information stream from one to many nodes concurrently. In the last few years, multicast routing has attracted a lot of attention from the network community, since many emerging applications are of multicast nature, such as teleconferencing, tele–education, and computer aided collaborative work. A multicast connection can substitute for many unicast connections carrying the same information, while reducing the network load.

The Internet is a packet–switching network that principally provides best–effort service. That is, there are no guarantees for services and applications running over it; applications may "starve", end–to–end delays may be arbitrary, and packets may be lost. Our work is motivated by the need to support QoS–sensitive multicast applications.

This document describes a QoS–aware multicast routing algorithm that implements resource reservation at the IP routing level. This approach aims to alleviate a problem that may be encountered with other frameworks (see RSVP [5]), under which routing is decoupled from resource reservation: the inability to satisfy receiver QoS requests even when suitable paths exist in the network.

There exist many other protocols for QoS multicast routing, like QoS–CBT, QoSMIC and QoS–PIM. QoS–CBT [2] extends the existing protocol proposed by A. Ballardie in [6] and supports one bi–directional shared tree. It was designed so because the primary objectives of the protocol were to maintain scalability while providing efficient link sharing. Consider a QoS–enabled scenario where multiple sources of a multicast group are transmitting through a shared bi–directional tree. Each sender has to transmit at a data rate that is the maximum of the individual data rates required by its receivers. In this case, a link on the tree may have to reserve bandwidth for packets from one or more of those senders. Based on the preceding reasons, source–specific trees seem better suited for scalable QoS–based IP multicast.

In [4], the original protocol QoSMIC that uses source specific trees is proposed. While this protocol is functionally feasible, the following problems can be identified. First, every new receiver join procedure requires an extensive network–wide search that is carried out by both the receiver and the multicast tree that it intends to join. Also, it increases significantly the join latency for a new receiver, because of the timeout set up for the reception of the messages generated by the candidate routers. Moreover, the path search–space

for a potential new receiver is restricted to the unicast paths used by the route advertisement packets, from the potential joining routers on the tree to the new receiver. This restriction may give rise to reduced join–success rate in certain scenarios.

QoS–PIM is an extension of PIM–SM [7] to support QoS traffic and it introduces two different policies for finding a feasible path. *TIQM* has scalability troubles because it tries to find a path basing on dynamic information about QoS parameters stored on each node. *NUQM* performs a search analyzing, once at a time, the paths returned from the underlying unicast protocol.

The first goal of this work is to increase the ratio of successful join the protocol can serve, and as a consequence the network utilization. QoSM^{2}P uses particular information to implement a search engine that is able to look for many feasible paths within the same join process.

Our approach is similar to that proposed in [8] to provide a QoS–aware unicast routing service in ad hoc wireless networks. However, the algorithm in [8] does not support multicast communications.

We want to give the user the possibility to specify more than one QoS metric without excessively increasing the complexity of the algorithm. Since the problem to find a bandwidth–constrained least–cost path in a network is NP–complete [9], we use an heuristic function to find a suboptimal path that allow us to limit the control overhead generated.

The paper is structured as follows: in section 2, we describe the proposed algorithm, different types of messages exchanged and data collected on the nodes. In section 3, we discuss the performance achieved by implementing the algorithm in the *ns–2* simulation framework. Section 4 concludes the work.

2. PROTOCOL OVERVIEW

The QoSM^{2}P protocol has been designed to support multimedia applications that generate large amount of traffic. Our purpose is to achieve a better usage of the Internet public infrastructure trying to serve as many users as possible at the same time. Then, our primary goal in the protocol design was the increase of the percentage of join requests accepted in comparison with other existing protocols like QoS–CBT and QoSMIC.

Since the response time observed by end users is particularly important for multimedia applications, another important parameter we considered is the join and leave latency. For example, registered users of a cable TV company have to switch among many different shows, causing frequent changes of the group membership. For this reason, another goal we would like to achieve is the reduction of the join and leave latency.

The Join procedure is initialized by a *designated router*[1] when it receives a request from a client directly connected. The DR then starts sending some messages containing particular information data called "*tickets*" (see section 2.1). QoSM^{2}P does not require the exchange of additional state information about QoS parameters at every change of the group membership. It takes decision about routing basing on data usually stored in the network nodes by a link state unicast algorithm.

Without generality loss, we use the bandwidth as an example of QoS metric for the rest of this paper. QoSM^{2}P is able to negotiate any number of different metrics using the same independent format for packets exchanged (see section 2.3). It permits the change of existing requests without disconnecting the client from a group.

The algorithm can be used to construct the distribution tree basing on given requirements specified by the connected clients and the source of the effective data flow. We use different types of messages, listed in Table 1, to support join, leave and change of existing requirements of a client.

Table 1: Message types

Type	Explanation
Join Req	A DR starts a network inquiry to find a feasible path for the connection
Join Ack	The Core accepts the request of the DR and stabilizes the path
Join NAck	The request has been rejected by the Core
ReDim Req	A DR or an on–tree router requests the change of the allocated resources
ReDim Ack	An on–tree router accepts the ReDim Request
ReDim NAck	An on–tree router rejects the ReDim Request
Leave	The message is forwarded upstream on the tree to allow the disconnection of a DR that has no more clients connected

Each packet type presents a common header containing the following data fields, needed to correctly manage the requests:

- *Type* Field used to discriminate between packet types
- *CID* IP address of the source of the data flow (i.e. the core router)
- *GID* IP address of the multicast group
- *NID* IP address of the designated router that initiates the request
- *RID* Request Identifier, that is a sequence number unique for each designated router

Since we want to build source based trees, we will refer to the source of the data flow as the *Core Router* of a given distribution tree.

2.1 Ticket information

We use the term "*ticket*" to represent an ideal token used to check the existence of a path suitable for the connection to the multicast distribution tree. These tokens are used either:

- to install a temporary reservation of the required resources that can be later confirmed;
- to limit the total number of different paths examined in a join procedure. Since every ticket can travel through a different path, we are able to control the diffusion of the network search varying the number of tickets enclosed in a join request message;
- to collect information about dynamic conditions encountered on the network nodes. These data can be

[1]The *designated router* is trivially the network node returned by the IGMP protocol, that is responsible for the join and leave of clients directly connected.

later used for the selection of the best path to install among all those examined;

Obviously the amount of tickets initially generated by a *designated router* affects the control overhead. In a real implementation our routers can be configured to start inquiring the network with a limited number of tickets and then, only if no suitable paths are found, to retry enclosing more tickets enlarging the diffusion of the search.

2.2 Data Structures

In order to guarantee the correct behaviour of the protocol, some data must be stored on network nodes. For instance, each router must store information about the available bandwidth on the configured network devices. Besides, it has to maintain the following data structures:

- `Installed paths`, stores data about *incoming* and *downstream* interfaces for every couple (Source, Group) - <S,G>. This table is used both for the routing of the active data flow and the check of resources already allocated at the arrival of a new request;
- `Temporary reservations`, collects information about Join and ReDim requests received. It contains the specification of requested resources, the identifier [2] and a flag indicating if the path has been already installed. This table is used to temporary reserve the bandwidth necessary for a connection when a join request arrives; the reservation can be later confirmed by an Ack message or deallocated by a NAck.

Other collected data differs among different types of router involved. A *Core router* has to collect data about the requests received in a table called `Requests database`, until it selects the best ticket to install. The *designated router* maintains a `Client database` with information about all the connected hosts in the local area network and the respective QoS requirements.

The storage needed to maintain these data structure, in presence of frequently changing groups, might reveal as a limit in a real implementation of QoSM^{2}P. At this stage of the work we have not yet considered a multi–domain or hierarchic architecture.

2.3 QoS Constraints

In order to contain the complexity of this exposure, we are making examples using one QoS constraint, typically the bandwidth. Most of existing protocols work specifying just one metric and have problems to scale well using two different metrics.

QoSM^{2}P has been explicitly designed to get round this problem, to allow any client to specify its QoS requirements using as many constraints as it needs. For this purpose, we have introduced into the join request packet format, a *Constraints* field. This is the sequence of specifications that the communication channel must satisfy. For example, a client can request a connection with a given bandwidth B, a maximum end–to–end delay D and a packet loss ratio to be not more than L.

In order to correctly manage asymmetric links, we assume that every node is able to maintain information about the "incoming" QoS parameters. In practice, we request nodes to have a measure of the performance that each directly connected peer can guarantee them. This can be achieved with an initial setup phase, in which every router of the network communicates to adjacent nodes its outgoing QoS parameters.

[2] A request can be identified by the couple $< \mathrm{NID}, \mathrm{RID} >$

2.4 Joining a group

When a client needs to join a specific group, it notifies the *designated router* of its local area network. The DR checks the presence of an entry in the `Client database` and, if the request has not been previously traced, generates a *Join Request* packet. This message consists of the common header and the following information fields:

Validity	Flag indicating if the nodes on which the packet has travelled so far satisfy the specification and have reserved the necessary resources
TTL	Time–to–live of the message. It has to be expressed in terms of real time, rather than hops count, in order to ensure the correct termination of the algorithm
T_{Tot}	Total number of tickets initially transmitted by the *designated router*
T_n	Number of tickets carried by the current packet
T_{ID}	Lowest identifier of the tickets carried
Constraints	Any sequence of QoS specifications to be satisfied (see section 2.3)
Costs	Information about the costs of the path examined so far

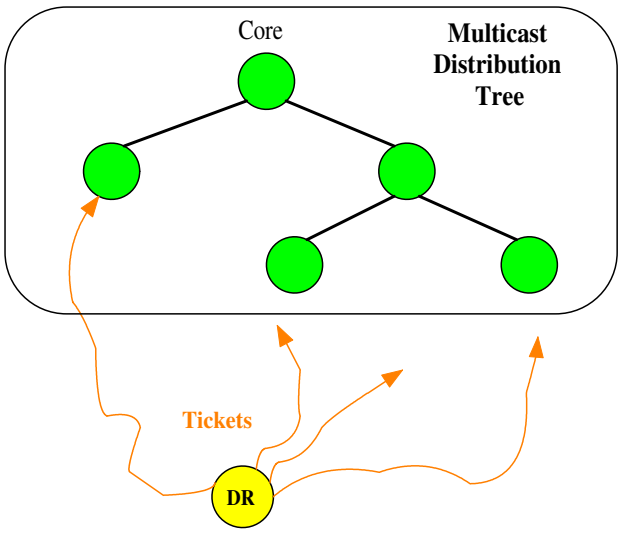

Figure 1: Example of Join process.

A Join Request message travels on the network and can be splitted into many pieces, each one passing through a different path. When a request carrying T_n tickets ($T_n \leq T_{\mathrm{Tot}}$) arrives, the router first checks the existence of an entry in the `Temporary reservations` table, to avoid problems about retransmission of packets and the possibility to generate loops in the distribution tree. If the couple <NID,RID> has not been previously inserted for the data flow <CID,GID>, it records the entry and performs the following checks:

- If the node is on–tree, it has to maintain an entry in the `Installed paths` table about <S,G>. The router will only check the availability of bandwidth on the incoming interface previously registered for that couple. It considers both resources already allocated and others possibly available for reservation, and:
  - if there are sufficient resources to satisfy the request, the router reserves the necessary bandwidth;
  - if the bandwidth is not enough, the router invalidates the tickets.

  Then, the message is forwarded to the upstream node in the tree.
- If the router is not on–tree, it checks the existence of interfaces with sufficient bandwidth among those from which the same join request has not been received before[3]:
  - if there are no suitable interfaces, it invalidates the tickets and sends a unicast message carrying them to the Core. We decided to forward invalidated tickets, instead of setting up a timeout on the core router , in order to minimize the response time observed by the end–user;
  - otherwise, it splits the tickets received basing on an heuristic function locally computed (see "*partition function*" in section 2.4.2) and forwards the messages generated to the selected adjacent routers.

The tickets are also used to collect in the *costs* field the dynamic information about QoS parameters encountered on the path. This information is stored on the Core that waits for a given amount of time, starting from the reception of the first message concerning a given Join request, the reception of all the tickets T_{tot} initially generated.

Eventually, the Core router decides for the request:

- if at least one valid ticket exists, the Core selects one of these, according to an heuristic function locally computed[4] (see "*selection function*" in section 2.4.2) and generates a *Join Ack* message;
- if there are only invalid tickets, the Core rejects the request and notifies the DR by sending a *Join NAck* message.

To optimize the ratio of successful join requests, a Join Ack message is sent on all network nodes, on which the tickets of the interested request have previously travelled. The Ack packet is used both to install the correct path and to release the unnecessary allocated resources. It contains, in addition to the common header, the number T_{ok} of the ticket selected for the path installation.

For the purpose of freeing the previously allocated resources in the case a request is rejected, a NAck packet is also sent on all the examined paths. The request identifier RID carried on the message is used to retrieve the correct entries to delete in the table of `Temporary reservations`.

[3]This check is done to avoid the possibility to create loops in the tree.

[4]The Core can select the ticket that has allocated the minimum amount of bandwidth on the network.

We have not chosen a "soft state" approach because it can bring to false rejections of join requests when the group membership often changes. In this case, a client requesting a connection for a given bandwidth can find the channel busy, because another host has temporarily reserved the resource, also in the case this one has already installed another path. This approach allowed us to maximize the ratio of accepted join requests, increasing the number of clients served and, consequently, the profits of a virtual ISP.

2.4.1 Timeout for reservations

In order to avoid that network faults[5] excessively penalize performances, we introduce the following timeouts on routers involved in the algorithm:

- τ_{res} is set on each network node at the arrival of a Join or ReDim request. After the expiration of this timeout, resources temporary allocated for a not yet confirmed request can be deallocated;
- τ_{core} is the maximum time spent by the core for the expectation of all the tickets of a given request.

2.4.2 Heuristic functions

In order to obtain an algorithm independent from the underlying unicast protocol and network architecture, we have defined two different heuristic functions that can be considered as "black boxes" and are:

- *Selection function*: used on the core router to choose the better path among those examined by the collected tickets;
- *Partition function*: computed on each router to split the tickets received among the candidate outgoing interfaces.

These functions can differ on the base of the requested constraint type and the reliability of information locally stored. The effectiveness of these functions impacts the efficiency of the final distribution tree. For instance, if the client wishes to have a low end–to–end delay, we will prefer to build distribution trees with higher breadth, trying to limit the depth.

The information collected by an underlying link–state unicast algorithm, can be used to compute the *Partition function*, basing only on the hop distance from the local router and the Core. Consider a node R_i receiving on the interface number 3 a join message carrying $T_n = 12$ tickets for the couple <CID,GID> and requesting 10 Mb/s of bandwidth. The router obtains by the underlying unicast (i.e. OSPF) the following distances from CID, for each interface:

Interface (if_j)	if_1	if_2	if_3	if_4	if_5
Distance (Δ_j)	3	4	8	6	2
Available BTWH (Mb/s)	6	12	10	7	15

We expect that, after having purged interfaces number 1 and 4, that have no sufficient bandwidth, R_i will send more tickets on the interface number 5 rather than on if_2. It can trivially partition the tickets received, sending T_k of these on interface if_k, using the following *partition function*:

[5]For example, the loss of a NAck packet does not allow to free useless reservations.

$$T_k = \frac{T_n * \frac{1}{\Delta_k}}{\sum_j \frac{1}{\Delta_j}}$$

Then R_i will forward the packets on if_2 and if_5 carrying:

Interface (if_j)	if_1	if_2	if_3	if_4	if_5
N. of forwarded tickets (T_j)	0	4	0	0	8

The *selection function* can be implemented differently varying on the goals of the network manager. If we want to preserve the network usage we will prefer to accept connections that allocate a minimal amount of bandwidth on the network. If we want to guarantee to our clients a low end–to–end delay we will install the minimal length path.

2.5 Leaving a group

A client that decides to leave a group has to notify the decision to the correct Designated Router. The DR checks in the `Client database` the existence of other hosts member of the specified group and, if it has no more receivers directly connected, generates a *leave* message that is forwarded to the router upstream on the multicast distribution tree.

The leave packet is recursively sent upstream, causing the disconnection of nodes that receive it, until a router (with other connected downstreams) stops the message. In this case, a ReDim request (see section 2.6) for the remaining bandwidth can be generated and sent to the upper node in the tree.

In Fig. 2 we show an example of leave process (on–tree nodes are shadowed).

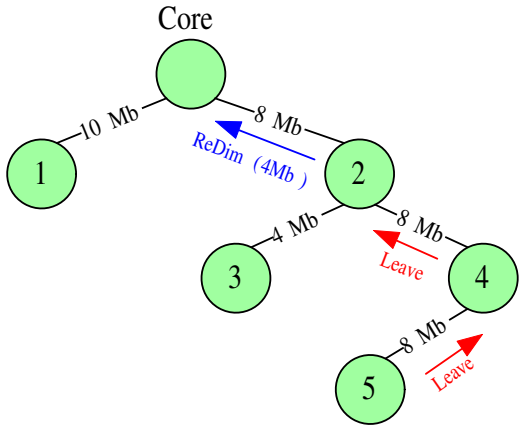

Figure 2: Example of leave process.

The DR 5 receives a leave request from a directly connected client and initiates a leave message that is forwarded upstream until it arrives in the node 2. Then, the on–tree router 2 checks the existence of another downstream and generates a ReDim Request specifying the residual bandwidth (4Mb/s). Fig. 3 shows the situation after the completion of the leave procedure.

2.6 Changing existing requirements

A *ReDim* packet encapsulates the new QoS specification for the communication channel and thus can be used for both increasing and decreasing the allocated resources. It can be generated when:

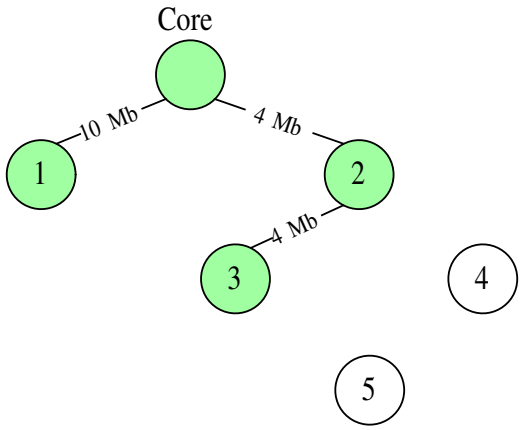

Figure 3: Situation after the completion of the leave.

- a client wants to join or leave a group with other receivers in the same local area network;
- a client explicitly requests the change of its QoS specifications;
- the leave of a downstream node on the tree causes a change of the existing requirements.

The first on–tree node with an incoming connection from the core that matches the new specification, can stop the forwarding of the ReDim packet and instantly notify the decision to the router that has generated the request.

To avoid the possibility of generating loops in the multicast distribution tree, we restrict the use of the ReDim messages, allowing the check of the needed resources only on the path previously installed on the tree, without looking for further alternatives. If the ReDim request is rejected (due to a lack of additional available resources on the path installed), a client can decide to leave the group and start another join request to cause a new wide path search.

3. SIMULATION RESULTS

We study several aspects of our protocol through simulations using the *ns–2* simulation framework. We have implemented a module that provide basic methods for the installation of the distribution tree for QoSM^{2}P and QoS–CBT and a simplified version of QoSMIC (using the "local minima" algorithm for the designation of candidate routers). We are reporting results obtained about the percentage of successful join requests, the control overhead and the multicast efficiency of generated trees.

We have generated two different network topologies with 64 nodes each, connected through links of 2Mbit/sec. One of these is a close–meshed network presenting a higher number of connections (about 6 outgoing interfaces for each node) than the other (3/4 links).

We instantiate a number of groups with a casual core router, transmitting a 1Mb/s data flow and we start 10 join requests for each group. The tests compare the measures among QoSMIC, QoS–CBT, QoSM^{2}P with only one ticket, QoSM^{2}P with 5 and 10 tickets.

We consider only the available bandwidth as an example of QoS metric. The output interfaces selected for the forwarding are those with enough available resources. As in

the example set forth in sec. 2.4.2, we have implemented a *partition function* such that the lower is the distance of the neighbor from the core, the higher is the number of tickets forwarded to it. In order to achieve a better usage of the network, we implemented a *selection function* that finds out the ticket that allocates the minimal amount of bandwidth.

3.1 Successful Join request Ratio

We have studied the amount of join requests that can be served, varying the number of groups created and the network load. In Fig. 4 we show the obtained results for the close–meshed network, while in Fig. 5 are reported the performances on the scattered topology. We can observe that, almost in every condition, QoSM^{2}P is more efficient than QoSMIC and QoS–CBT, finding more possible paths and then satisfying a greater number of join requests.

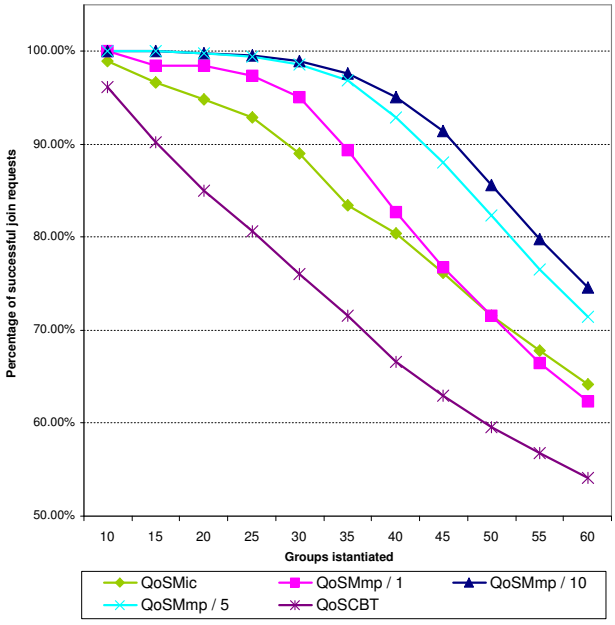

Figure 4: Percentage of successful join requests (close–meshed network).

QoSM^{2}P is able to satisfy more requests than QoS–CBT, even with only one ticket, because it performs the check on resource availability on all output interfaces, rather than being constrained to the interface provided by the unicast routing service.

QoSMIC and QoS–CBT can fail the join also in relatively good network conditions, due to the limits exposed in the introduction. We can see that with only 10 groups present (it can be better observed considering the first network topology), QoSM^{2}P is able to satisfy all the requests (99.9%) generated versus the 93.8% accepted by QoSMIC and the 89.9% or QoS–CBT.

We can notice that augmenting the network load, instantiating more groups, the ratio of QoSMIC exceeds the one of QoSM^{2}P / 1 ticket but remains considerably lower than QoSM^{2}P with 5 or 10 tickets.

Increasing the network load, the difference between the version of QoSM^{2}P with 1 ticket and the one with 10 tickets increases, showing the advantages of a wider research of

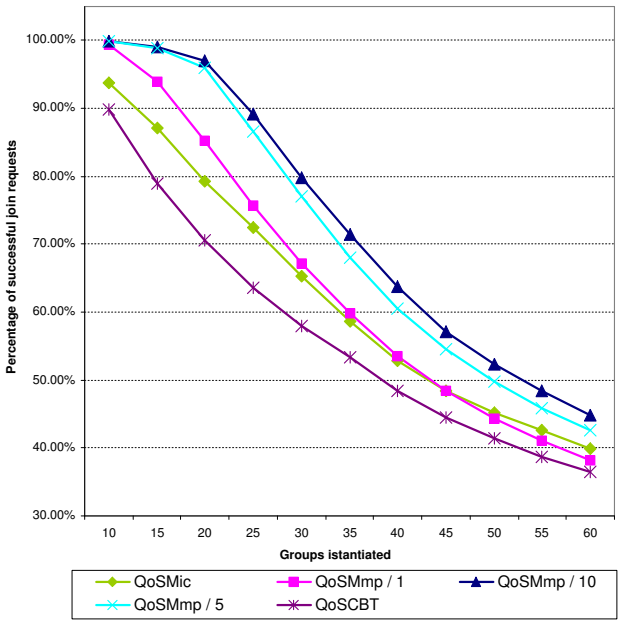

Figure 5: Percentage of successful join requests (scattered network).

possible paths. Generating a great number of join requests (60 groups instantiated on the close–meshed network), we have obtained very good performances for the version with 5 and 10 tickets, satisfying respectively the 71.38% and the 74.62% of the total, while the version with 1 ticket performs more like QoSMIC arranging around the 62/64% of the total.

3.2 Multicast efficiency

The "*Multicast efficiency*" *ME* can be useful to give a measure of the goodness of the multicast distribution trees built with an algorithm. It is obtained as:

$$ME = \frac{\sum_{\text{members}} (\text{distances from core})}{\text{Total number of links allocated}}$$

We have surveyed the measures varying the number of routers requesting the join to 3 different groups, transmitting a data flow of 1Mb/s each, on a close–meshed network with 2Mb/s links. In Fig. 6 we show the average results obtained for the three groups instantiated using QoS–CBT, QoSM^{2}P with 1, 5 and 10 tickets.

The usage of tickets in the setup phase of the QoSM^{2}P algorithm allows us to find paths that allocate less bandwidth for the connection of new routers. More tickets used in the setup phase can guarantee a better performance in the distribution of the actual data content.

3.3 Control overhead

Finally, we report in Fig. 7, the statistics about the *control overhead* of setup messages, generated comparing QoSM^{2}P and QoS–CBT on the scattered topology.

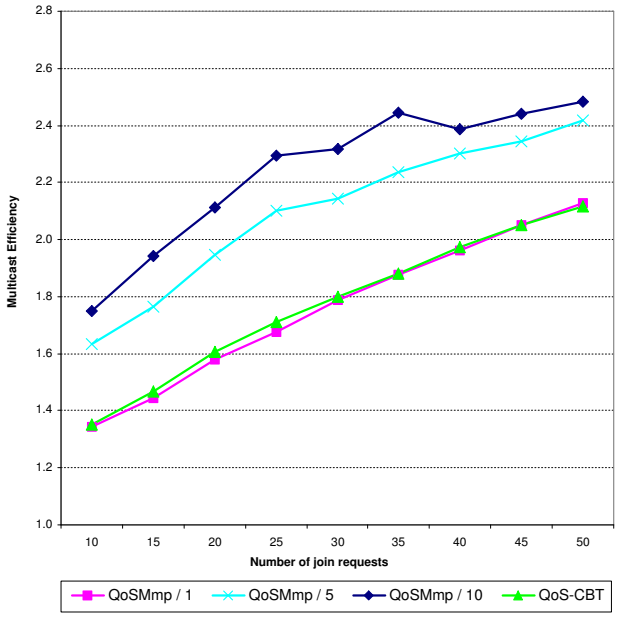

Figure 6: Multicast efficiency.

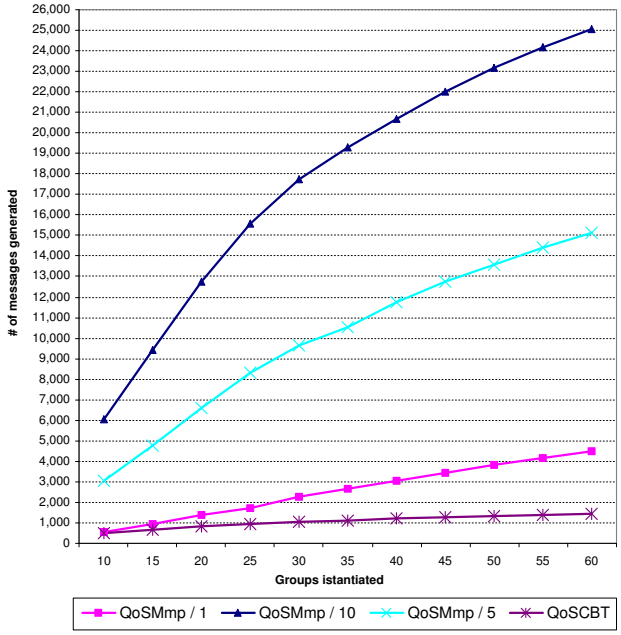

Figure 7: Control overhead.

The control overhead generated by QoSM^{2}P can be approximated as:

$$\text{Control overhead} = \text{O}\ (T * N * G * K)$$

where T is the number of tickets initially enclosed in the join request, N is the number of groups instantiated and G the group density; K is a constant varying with the network topology.

We have noticed a significative increment of the number of messages exchanged, compared with that of QoS–CBT. Nevertheless, the packets are very small and do not consume a large amount of bandwidth travelling hop–by–hop. We think this is an acceptable arrangement if we want to increase the ratio of successful join requests accepted, the number of served clients and consequently augmenting the network utilization.

4. CONCLUSIONS AND FURTHER WORKS

We propose QoSM^{2}P , a protocol for supporting QoS–sensitive multicast applications over the Internet. This protocol identifies multiple paths in the same Join Process, and selects the most promising one using dynamic information that the tickets can collect on the travel.

The decentralization of the computation on the various routers involved in the protocol has significantly improved the performance of the Join Protocol, augmenting the ratio of successful requests we can serve and the multicast efficiency of the distribution trees.

The most important problems arising with the usage of QoSM^{2}P in presence of high density groups is the amount of memory needed to store information in `Temporary reservation` and `Installed path` tables. Further works will focalize attention on the design of a hierarchical infrastructure in order to allow a better scalability for sparse and dynamic groups.

We are still evaluating the possibility to extend the algorithm for the creation of a bidirectional shared tree. Many other protocols (like [3], [4]) use this approach to preserve the resources on network and routers, creating first a connection to a single bidirectional tree and then, only if the application presents particular needs, a source centered tree.

5. REFERENCES

[1] Pierpaolo Baccichet. *Analisi e valutazione di protocolli di instradamento multicast multi–path per traffico con QoS*. Master Thesis, May 2002.

[2] J. Hou, H. Y. Tyan, B. Wang, Y. M. Chen. *QoS Extension to CBT*. Internet draft, March 1998.

[3] S. Biswas, R. Izmailov, B. Rajagopalan. *A QoS–Aware Routing Framework for PIM–SM Based IP–Multicast*. Internet draft, June 1999.

[4] R. Pankaj M. Faloutsos, A. Banerjea. Qosmic: Quality of service sensitive multicast internet protocol. *Proc. ACM SIGCOMM*, September 1998.

[5] L. Zhang, R. Braden, S. Berson, S. Herzog, S. Jamin. *Resource reSerVation Protocol (RSVP) - Version 1 Functional Specification*. RFC 2205, September 1997.

[6] Z. Zhang A. Ballardie, B. Cain. *Core Based Tree (CBT ver. 3) Multicast Routing - Protocol specification*. RFC 2189, September 1997.

[7] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering. *Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification*. RFC 2362, June 1998.

[8] S. Chen, K. Nahrstedt. *Distributed Quality–of–Service Routing in Ad Hoc Networks*. IEEE Journal on selected areas in Communications, August 1999.

[9] D. Johnson M. Garey. *Computers and Intractability: A Guide to the Theory of NP–Completeness*. 1979.