Principles of QoS in Group Communications

L. Mathy C. Edwards D. Hutchison*

Computing Department, Lancaster University, Lancaster LA1 4YR, UK
E-mail: {laurent, ce, dh}@comp.lancs.ac.uk

In this paper, we consider QoS support in the context of group communications. In particular, we present fundamental properties of QoS in group communications, which, although quite simple, have been widely overlooked and/or misunderstood. Because distributed multimedia applications require flexible QoS agreements, end-to-end QoS negotiation will play an important role. We show how such a negotiation mechanism can be designed, taking the fundamental QoS properties into account. This leads us to the design of an algorithm for QoS negotiation whose computational overhead proves to be independent of the size of the group of receivers.
Keywords: Quality of Service (QoS), Group Communication, Consensus QoS Parameter, Receiver-dependent QoS Parameter, QoS Negotiation.

1. Introduction

As multimedia applications develop, communications systems are expected to make provision for two of their aspects, namely Quality of Service (QoS)support and group communications services. Although these two topics have now been studied for some time, the proposed solutions were too often either strong on QoS and weak on group communications [5,16] or the opposite [8,12]. As a result, current communications systems do not provide strongly integrated support for QoS and group communications simultaneously. Because group communication problems have long been better understood than QoS issues and also because multicast has the obvious potential to optimise resource utilisation, group communications have become far more ubiquitous than QoS support. Consequently, to cope with the lack of QoS support while making ef-

* We are grateful to the UK Engineering and Physical Sciences Research Council for supporting Laurent Mathy and Christopher Edwards, and to BT Labs for the CASE sponsorship of C. Edwards' PhD studentship.

ficient use of group communications, applications that are able to adapt their performance requirements have been developed. To do so, these applications adjust their communication needs to the conditions in the communication system and are therefore said to be *adaptive*. However, even the best adaptive techniques are powerless when facing the poorest conditions in the communication systems. On the other hand, in order that distributed multimedia applications become ubiquitous, especially in a commercial environment, there is an urgent need for communications systems to be able to support applications that require more control and better guarantees over their performance. Indeed, for applications such as tele-teaching, tele-conferencing, video on demand and simulations, especially in a commercial context, starting a communication session may be worthwhile only if some minimal performance can be guaranteed throughout its duration. Such applications can be collectively called *critical applications*.

In this paper, we present fundamental properties of QoS in group communications, which, although quite simple, have been widely overlooked and/or misunderstood. We believe that the correct understanding and use of these properties by communications systems designers will be one of the key factors in deploying effective integrated multimedia communications systems able to support critical applications. Also, we show that, because multimedia applications, and in particular critical ones, require flexible QoS agreements in order to achieve optimal communication resource utilisation, end-to-end QoS negotiation will play an important role. We study the impact of the fundamental QoS properties on the design of such a negotiation mechanism.

The paper is structured as follows. Because our results are very general, we try to avoid targeting them towards particular communication environments, and we present our own simplified communication architecture in section 2. Section 3 describes fundamental properties of QoS in group communications. In section 4, we review related work and show how our fundamental results fit into it. Then sections 5 and 6 show how to design an end-to-end QoS negotiation mechanism and associated algorithm, taking into account the properties of QoS described in section 3. Section 7 records the lack of appropriate support for QoS in group communications as provided by some of the major network technologies, and consequently suggests directions for further research. In section 8, we present simulation results of the method proposed in section 6. Finally, section 9 concludes the paper.

2. Communication Semantics

There are numerous different semantics for data communications. These semantics can usually be described in terms of a set of properties clearly characterising the communication. For instance, it is obvious that a communication where data is delivered in sequence and without loss or error has different features from one where some data may not be delivered to the recipient. However, although explicitly specifying the properties of a communication is the safest way to describe it, usually it is also very cumbersome. Therefore, over the years, "labels" have emerged to identify some of the communication semantics. For instance, two of the most widely used "labels" are *connection-oriented* and *connectionless* data transfer.

Meanwhile, different communication environments, and associated "schools of thought", have emerged as well, each developing its own jargon. The result is that, often, some words (i.e. labels) are interpreted differently by different people or even given different meanings altogether. It is interesting to ask people what a connection is, for example, and to find oneself with a handful of definitions each of which, unfortunately, makes sense!

Therefore, in order to avoid any confusion and to stay as general as possible, we briefly expose here the different abstract communication concepts that are of interest in the remainder of the paper. By "abstract", we mean that our discussion is kept independent of any detailed considerations about how these concepts may be put into practice in different communication environments.

2.1. A General View of Data Communications[1]

To be as clear as possible and without loss of generality, it will be sufficient for us to consider the communication systems as being composed of two major parts: a set of *end-systems*, where data is explicitly produced and/or consumed, and a *network*, responsible for moving data between or among end-systems.

Because of the multimedia communication context considered in this paper, we may also require to be able somehow to instruct the network to perform its task within given *traffic constraints*. When necessary, we will consider a more "detailed" view of the network as being composed of a set of *relay entities*, somehow

[1] In this section, our intention is not to give rigorous and precise definitions of network architecture components, but rather to fix ideas in an informal way.

linked and co-operating with one another in order to achieve a routing/forwarding task.

An end-system will be seen as composed of three hierarchical components or layers. The first of these layers is dependent of the network technology used and can also be considered part of the *network*.

The second layer, which performs common communication tasks, also provides the layer above it with a common *interface* (boundary between two components or layers) while shielding it from the details of the network technology in use. Because this layer "hides" the network, it provides for virtual communications with layers of the same level at other end-systems. For this reason, this sort of communication is said to be *end-to-end*, and so is the layer. Again, the layer may be instructed to perform its tasks within given constraints. For convenience, this layer will be called the *transport layer*.

Finally, the third layer we consider is where the data is actually produced and consumed. It is of course another end-to-end layer and will be called the *application layer*. Because the application layer is the one aware of the data semantics, it is also the one able to formulate the constraints under which data should be manipulated by the transport layer and the network. The application layer represents the communication-specific part of an application. The other parts of the application reside above the application layer.

When considering an interface, we will call the set of components beneath it a *communication sub-system*. The global functionality provided (at an interface) by a communication sub-system will be referred to as a *service*, while the way the service is realised will be called a *protocol*. The communication sub-system is then a *service provider* while the components using the service can be called *service users*. A service user and the service provider interact with one another (at a service interface) by exchanging *service primitives*.

Finally, we give the name *Quality of Service (QoS)* to the constraints expressed at an interface.

Again, while such a model, illustrated in figure 1, does not correspond to one particular communication architecture, it encompasses each of them and fixes the ideas about the vocabulary used in the rest of the paper. For instance, if we consider the Internet protocol suite, the Internet Protocol (IP) is at the network layer of our architecture, while the TCP and UDP protocols are at the transport level. Finally, the RTP protocol fits the semantics of our application layer.

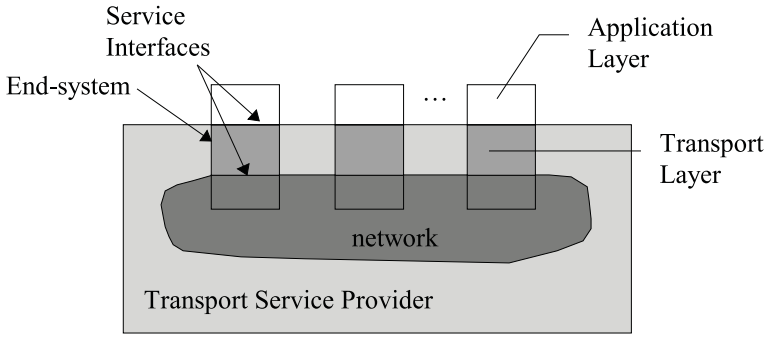


Figure 1. Simplified communication architecture, showing three end-systems connected to the network.

2.2. Communication Semantics

The most general model of a communication comprises 3 phases:

1. set-up phase;
2. data transfer phase;
3. release phase.

It is possible for the set-up and release phases to be omitted. For any phase carried out, the way the phase is performed and the rules enforced during the phase may give rise to different communication semantics. For instance (these examples are not exhaustive), interactions in phase 1 can involve a single service user and the service provider or it can involve an exchange of set-up messages between a set of service users;the set-up could also be implicitly achieved (see below). As for the release phase, it can be either explicit (with exchange of service primitives) or implicit (without service primitives); also, the rules enforced during the data transfer phase can require the service provider to ensure an in-sequence, error-less delivery of the data or not, or timely delivery. There are numerous other examples of possible communication semantics.

From the most general point of view, the use of given communication semantics at any level of the architecture should not preclude the use of other semantics at other levels (unless clear technical evidence dictates so).

The important point for us is that if a set-up phase takes place (whether explicitly or implicitly[2]), all data items sent in the resulting communication can then be *related* to each other. In other words, the data can be marked with some identifier which, in its most general form, is the triplet (source ID, destination ID, semantic information) where the source and destination IDs identify the sender and receiver(s) of the data respectively; the semantic information is some form of tag, established during the set-up phase, unambiguously identifying the corresponding communication. The tag is needed when several simultaneous communications can be established among the sender and the receiver(s). Such an identifier thus establishes a *context* relating the data items together.

We will be particularly interested in the case of communications where the set-up phase is performed explicitly and involves three parties: a sender, the receiver(s) and the service provider. Indeed, it is precisely in such a case that the protagonists are given the chance to negotiate various characteristics of the communication, including the QoS.

This slight digression allows us to avoid the use, in the rest of the paper, of potentially misleading terms such as "connections". Furthermore, the concepts and techniques presented in the remainder of the paper can easily be adapted to different communication semantics.

3. Quality of Service and Group Communications

In this section, we review the major QoS parameters used to specify the characteristics of the data transfer phase of group communications, as seen at service interfaces. This section describes general communication characteristics that are independent of any layer in a communication architecture.

3.1. QoS Parameter Analysis

3.1.1. The Throughput

The throughput is a measure of the rate at which data is generated. Although different semantics can be found in the literature (e.g. instantaneous throughput, average throughput, peak throughput, etc.), the throughput is mea-

[2] An example of implicit set-up occurs in the case of UDP communications where the header information used throughout the lifetime of the communication is implicitly agreed between source and destination.

sured in bits per seconds (bps) and expresses communication capacity. For traffic having tight communication requirements (e.g. multimedia traffic), the throughput is one of the key performance QoS parameters.

It should be noticed that because it is a rate, a throughput can only be defined (and measured) on semantically related pieces of data. In other words, the throughput is a performance QoS parameter defined in a context where a set-up phase was performed before data transfer occurs in the communication.

A piece of data, emitted at a service interface, is conveyed through the communication sub-system as one piece or several consecutive smaller pieces of data which, for simplicity, we call packets[3]. Each of these packets will either reach (some of) the end-systems where receivers reside allowing the reconstruction of the corresponding data which can then be successfully delivered, or it will be lost in the communication sub-system. These losses are usually caused by buffer overflows in network relay entities and end-systems.

If the rate at which pieces of data are produced (i.e. the sender throughput) is higher than the rate at which they are consumed by one or several receivers (i.e. the receivers' throughput), the communication sub-system has either to store somehow the associated packets for later delivery or else discard some packets. If such a situation persists, the stored packets will eventually consume the resources (mainly buffers) allocated or available to the communication. This results in congestion and causes uncontrolled losses of packets, and thus incurs data losses. Because the capacity bottleneck can occur anywhere in the communication sub-system, communication resources are wasted during periods of congestion: the closer to the receiver the congestion occurs, the more resources are wasted.

Point-to-Multipoint Communications

For group communications with a single sender (point-to-multipoint communications), we therefore see that, in order to avoid a sub-optimal situation where communication resources are wasted due to congestion, a receiver throughput should only be temporarily lower than the sender throughput. Also, if the communication requirements allow it, the sender can be forced to slow down through the use of congestion avoidance and/or flow control mechanisms.

Now, let us suppose that after a period where the sender was sending faster

[3] This is a simplified view, for the sake of clarity, that does not damage the generality of the discussion.

than the receiving rate of some receivers, these receivers are catching up by consuming data at a rate higher than the sender throughput. Clearly, such a situation cannot last indefinitely: eventually, there will not be any more "excess" data stored in the communication sub-system and the receivers will then slow down to the sender throughput.

We therefore see that indeed a receiver throughput can only be temporarily higher than the sender throughput.

Consequently the throughputs at the sender and at the receivers are not independent and their nominal values, as well as associated communication resources, should be the same in order to ensure a correct and optimal use of the communication sub-system. Such parameters, whose value must be the same for every member of the communication, are called *consensus QoS parameters.*

The point is that, to avoid a sub-optimal use of resources as well as uncontrolled losses, the same "capacity" must be uniformly granted to a flow of data throughout the communication sub-system: the de-coupling of the receivers' capacity with the sender's capacity as allowed, for instance, by "IP multicast" can only be achieved to the detriment of optimality. Finally, the reader should note that the goal of adaptive applications is to adapt their rate to a consensus throughput which is not known beforehand and which fluctuates in time.

Multipoint-to-Multipoint Communications

In group communications where there can be several senders (multipoint-to-multipoint communications), we no longer have a unique throughput value to characterise the communication. Indeed, because we can have simultaneous sources of traffic (i.e. simultaneous senders), the rate at which data is injected in the communication sub-system is the aggregation (i.e. the sum) of the throughputs measured for each sender individually. In order to avoid the congestion problems described earlier in this section, each receiver must consume data at a rate matching this aggregation of individual throughputs. As a consequence, the senders and/or the receivers have to adjust their throughput whenever the number of senders varies.

In the rest of the paper, we will focus on point-to-multipoint communications, because of the uniqueness of their consensus values. It should however be clear that our results can readily be extended to the multipoint-to-multipoint case by properly modifying the sender's and/or receivers' consensus values.

3.1.2. The Transit Delay

The transit delay (or delay in short) is usually defined as the elapsed time between the emission of a piece of data by the sender and its consumption by a receiver. It expresses time spent in the communication sub-system[4].

Such a parameter, also called latency by some authors, is valid for independent single pieces of data as well as for related data.

When moved from the sender to the receivers, data passes through several components of the communication sub-system which are situated in the end-systems as well as in the network. Each of these components can be characterised by a processing speed and the size of buffers where data awaiting processing is kept. In a sense, the concatenation of the resources used by the data in each component of the communication sub-system can be seen as a distributed buffer.

Because the network paths to different receivers are different and because different receivers will possibly reside in end-systems with different capabilities, the size of the distributed buffer representing the communication sub-system between the sender and each receiver will vary from one receiver to another.

The longer the "equivalent" buffer is (i.e. the greater the "distance"), the longer it takes for a piece of data to cross it. Therefore, the delay measured by different receivers will be different. Because the values of the delay are mostly significant for the receivers, they do not have a significant influence on the behaviour of the sender as well as on the other receivers which appear as independent from a transit delay point of view. Therefore, the nominal values of the delays measured by different receivers can be different.

Such parameters, whose value can be different from one receiver to another, are called *receiver-dependent QoS parameters.*

3.1.3. The Delay Jitter

The delay jitter (or jitter in short), is usually defined as the maximum variation allowed for the delay suffered by pieces of data. It is measured in seconds and expresses the distortion inflicted on the data delivery pattern, or the degradation of the "temporal" quality of the data.

Because the jitter only makes sense when measured on a sequence of data,

[4] A communication sub-system, by our definition in section 2.1, includes parts of the end-systems. For example, transport layer buffers are part of the communication sub-system used by the application layer

this parameter is only valid for a related set of data. Any factor influencing the delay also influences the jitter. We therefore conclude that the jitter is a receiver-dependent QoS parameter.

3.1.4. The Error Rate

Error rate is defined as the ratio of incorrectly received (or lost) data to the sent data, and is therefore a dimension-less value. As for the throughput, different semantics are available (e.g. corruption rate, loss rate, bit error rate, packet error rate, etc.). An error rate expresses the degradation of the accuracy of the content of the transferred data.

Again, such a parameter needs to be measured over a sequence of data and is therefore only valid for related data. We therefore see that the existence of a context semantically relating the data sequence does not imply error free delivery of the data (although the reverse is true).

The error rate can also be interpreted as a measure of the degradation of the "spatial" quality of the data. Because the required quality, or indeed the perceived quality, can be different from one receiver to another, the error rate is a receiver-dependent QoS parameter.

3.1.5. The Degree of Reliability

The degree of reliability is defined as the minimum number of receivers that must receive each data item. It is measured as a number of receivers and expresses a criterion assessing the success (or failure) of a data transfer.

The definition of this parameter comes from the need to provide more flexibility than the "all or nothing" semantics which naturally define the success of data transfer in point-to-point communications. Used in conjunction with the error rates defined in the previous section, this parameter allows powerful (spatial) quality control.

However, because of its definition, the degree of reliability is directly relevant only to the sender. On the other hand, the mechanisms that will possibly be needed to enforce a minimum degree of reliability will influence the behaviour of both the sender and the receivers. The degree of reliability is therefore a consensus QoS parameter.

3.2. Classification of the QoS Parameters

In the previous sections, we have seen that in point-to-multipoint communications, the presence of the group of receivers introduces a new dimension, and therefore an extra degree of complexity, into the communications. From a QoS point of view, this new dimension results in the classification of the parameters into two categories:

- The QoS parameters whose scope is the whole point-to-multipoint communication, and whose values therefore affect the sender and all receivers. Such parameters are called *consensus QoS parameters*.
- The QoS parameters whose scope is limited to each receiver separately, and whose values can therefore be different from one receiver to another. Such parameters are called *receiver-dependent QoS parameters*.

This classification results from the very nature of the different QoS parameters and is therefore an intrinsic property of QoS in group communications. Table 1 summarises the classification of the QoS parameters that we have considered.

Table 1
QoS Parameter Classification

Consensus	Receiver-dependent
Throughput	Delay
Reliability	Jitter
	Error rate

4. Dealing with Heterogeneity

In the previous section, the concept of consensus QoS parameters was introduced and discussed. Although such parameters are a "fact" rather than a "choice", they may seem at odds with the heterogeneity usually encountered in group communications. In this section, we study the relations between consensus parameters and heterogeneity.

4.1. Related Work

In group communications, because the receivers can be spread across a wide-area network and/or reside in different types of end-system with different types

of access to the network, they can experience very different communication or processing capabilities. Furthermore, different receivers can also have different requirements as to the perceived quality of their reception. Much research has been dedicated to finding ways of dealing with such heterogeneity issues.

4.1.1. Filtering

One of the main techniques developed for this purpose is *filtering* [17]. Filters are entities that operate on encoded, related data in order to adapt the data flow to receivers' requirements. By strategically placing filters on the data path, the throughput, and consequently the quality, of media data flow can be optimally matched to the requirements of the different receivers.

At first glance, because of the individual adaptation to each receiver's requirements, the concept of filtering seems to be able to change the consensus nature of the throughput of a point-to-multipoint communication (see section 3.1.1). However, a closer look at this technique causes no disagreement with the immutable characteristics of our QoS classification.

Indeed, in order to work properly, that is to avoid uncontrolled losses, the rate at which filters at a given filtering stage must be able to treat data has to match the rate at which the same data is forwarded by the previous filtering stage. This also implies that the throughput at which a filter sends must be matched by the throughput at which the filters of the next stage of filtering receive. This is consistent with the characteristics of a consensus parameter.

Whether by directly accessing the data [17] or through the use of information tagged to it [7], filters operate on encoded data and therefore depend on the coding scheme used. In other words, their operation depends on the semantics of the data and consequently on the applications using it.

Because of this dependency on the data semantics, we argue that trying to develop filtering techniques that could be used in network relays (that is operating at the level[5] where routing/forwarding is performed) is not a correct approach. Indeed, by consuming processing power in the network relays, the filters would then undoubtedly reduce the effective bandwidth that could otherwise be achieved and also very much complicate the scheduling algorithms needed for the network to be able to commit to any kind of QoS support. Furthermore, developing

[5] We insist that the term "network relays" should be used to identify entities relaying information at the network layer level *only*

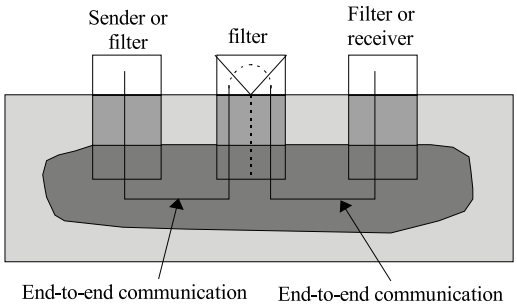


Figure 2. Filters as application gateways.

network relays specialised in dealing with a given type of information (or even worse, with a given type of coding for a given type of information!) seems to be a setback to the deployment of integrated networks: filtering network relays would be a step backward towards specialised networks.

As a result, we think that filters should be seen and used as application relay entities (or application gateways), as depicted in figure 2. By doing so, the specificity introduced by the use of filters is dealt with at the application level, thus allowing the network to retain its desired functionality, independent of the semantics of the data carried.

From this point of view, communications between filters are therefore end-to-end communications (from the transport layer semantics, at least), and we conclude that the consensus semantics of the throughput is thus retained. Obviously, the use of filters also results in a loss of end-to-end semantic information between the initial producer (the sender) and the final consumers (the receivers) of the data. Actually, on either side of a filter, both the transport and network may be different.

4.1.2. Hierarchical Coding

Another technique to accommodate heterogeneity in group communications is hierarchical coding of the data (e.g. see [15]). Here, the information is coded across several layers of representation, each layer adding accuracy (and thus quality) to the previous one. By selecting an appropriate set of layers of representation, receivers can then tune the quality they receive to their capabilities or needs. In hierarchical encoding, each layer of coding naturally represents a sequence of related data communication. Of course, in each of the individual communications,

the consensus nature of the throughput is obvious. Hierarchical coding thus deals with heterogeneity without any loss of end-to-end communication semantics.

Finally, the tagging of data can also be seen as a hierarchical coding scheme, where each value of the tag identifies a given "sub-communication", which are multiplexed onto a unique "physical" communication. In other words, such a technique, as well as the one consisting of using different priorities for different parts of a communication, simply splits the original flow of data into several "sub-flows", each of which is characterised by a consensus throughput.

4.2. Negotiation of Consensus Parameters

From the previous section, it should be clear that the consensus nature of some of the QoS parameters is indeed immutable. So far, consensus parameters (mostly the throughput) are established on a "take-it-or-leave-it" basis (e.g. see [1]). This is probably because of the difficulty of finding the final consensus values among the members of (sometimes large) groups. However, in many situations, QoS negotiation provides for a much-valued flexibility and turns out to be a very useful optimisation tool. For instance, in the filtering scenarios considered above, QoS negotiation among the filters at each stage of filtering would allow the installation of the most appropriate filters and could lead to fewer filtering stages. The same can be argued for the different layers of a hierarchical coding schemes.

Furthermore, because the choice of optimal end-to-end consensus values among the entities taking part in the communication also allows optimal resource set-up in the network, negotiation of consensus values is important in order to achieve optimal resource usage. We therefore see that the negotiation of end-to-end QoS values is a very important tool in heterogeneous environments, complementing other techniques by providing them with common support that they can use to achieve optimal settings.

Finally, QoS negotiation is a central mechanism in environments where QoS values represent contractual agreements between the communicating entities and the communication sub-system (e.g. guaranteed QoS, etc.).

5. QoS Negotiation in Point-to-Multipoint Communications

Receiver-dependent parameters, because their values are independent from one receiver to another, are negotiated as point-to-point values and cause no

particular problems. On the other hand, because of the difficulty of reaching consensus values with some QoS semantics, negotiation of consensus values may be more problematical. We therefore focus on consensus parameters, studying their negotiation for two very general types of QoS semantic: single values and interval values.

5.1. Common Grounds

In the following, all the negotiation schemes are shown as being initiated by the sender of the point-to-multipoint communication. There is no loss of generality in doing so, because these schemes can easily and readily be adapted to situations where the initiator of the negotiation is a receiver or even a third party (e.g. QoS broker).

QoS negotiation is ideally a triangular mechanism, meaning that it should involve three parties: the sender, the service provider (the communication subsystem) and the receivers.

The sender supplies proposed values for the parameters, based on the data semantics, to the service provider in an establishment (set-up) request service primitive.

The service provider then tries to deliver an establishment indication service primitive to each receiver. In these indications, the proposed values may have been modified by the service provider, in accordance with the QoS semantics, to reflect the service provider's capability to serve the receivers. Therefore, because these capabilities depend on conditions in the underlying networks (e.g. access speed, path, etc.), protocol implementations, end-system technologies and so on, the values may be different from one indication primitive to another (i.e. from one receiver to another). For the same reasons, it may also happen that the service provider is not able to offer the requested service to some receivers, whatever the (allowed) values of the QoS parameters may be. In such cases, the receivers concerned are simply "ignored" by the service provider and are not issued with an indication service primitive.

On reception of an indication primitive, a receiver can either accept or reject taking part in the communication. If it accepts, the receiver selects values for the consensus parameters based on both local capabilities (e.g. available buffer space, CPU allocation, etc.) and quality requirements, and returns these values in an establishment response primitive.

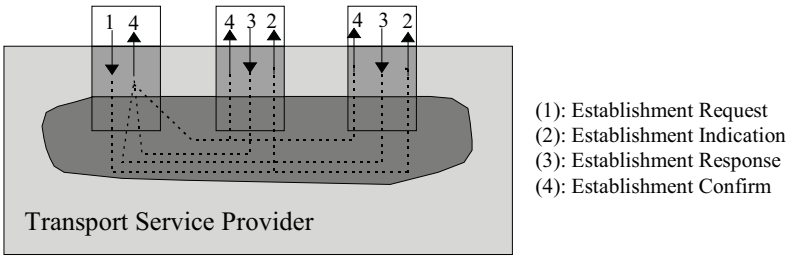


Figure 3. Three-way handshake.

Of course, if the receiver has to reject the communication (i.e. does not wish to or cannot take part in the communication), it issues a service primitive expressing this and is "ignored" for the rest of the negotiation.

At this stage, the service provider has collected the QoS requirements of the receivers that wish to take part in the point-to-multipoint communication. The task of the service provider is now to select consensus values for the consensus parameters (based on the values returned by the receivers). The selection of these consensus values is the subject of the next two sections.

When the service provider (usually a protocol entity at the sender's side) has reached the consensus values, these are advertised to all the users (sender and receivers) that are to take part in the point-to-multipoint communication, through the use of an establishment confirm service primitive.

For reasons that will be explained in section 5.3, the set of receivers that receive the confirm primitive (and that will actually take part in the communication) may only be a sub-set of the receivers that had positively responded with a response primitive, the rest of them having been issued a release primitive. We therefore see that the negotiation of consensus QoS parameters in point-to-multipoint communications requires a three-way handshake, illustrated in figure 3, in order for the service provider to confirm final adherence to the receivers.

5.2. Negotiation of a Single Value

A single-valued QoS parameter is characterised by a unique value. For example, peak throughput and mean throughput are both single-valued.

Finding the consensus value for such a parameter is not difficult: it is the weakest or the strongest of the values collected by the service provider, depending

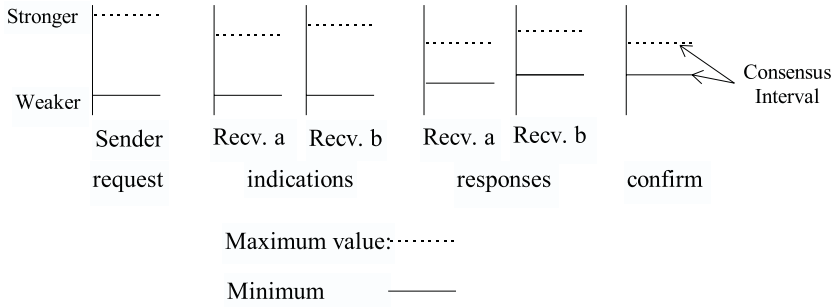


Figure 4. Rules for interval values negotiation.

on the semantic of the QoS parameter. For example, if the parameter represents a minimum requested value, the consensus (final) value is the strongest proposed one, whereas if the parameter represents a maximum allowed value, the consensus value is the weakest.

This type of negotiation will also be used in best-effort cases, where the negotiated values represent indications of the desired performance, but do not constitute any contractual commitment.

5.3. Negotiation of Interval Values

An interval-valued QoS parameter is characterised by a pair of ordered values, usually expressing lower and higher bounds on the parameter. For example, a classical min-max, a compulsory value [6,10] or an average-peak pair of values are all interval-valued.

Because interval-valued parameters can be seen as a pair of single value, their negotiations may, at first glance, seem quite straightforward. Indeed, because we can assimilate a lower bound to a minimum single value and a higher bound to a maximum single value, the consensus lower bound is the strongest lower bound collected, while the consensus higher bound is the weakest higher bound collected (see figure 4). Unfortunately, the negotiation of an interval-valued parameter may not be as direct. Indeed, recall that the service provider may change the values proposed by the sender, in order to reflect its capabilities. In the case of an interval value, we expect such a modification of the values to be a decrease[6] of the higher bound. In general, service providers will be happy to keep the lower

[6] In this section, we assume that the relation *lower bound* $\leq$ *higher bound* is respected in any service primitive.

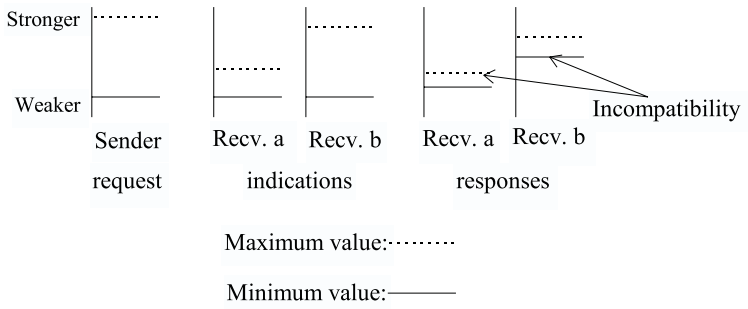


Figure 5. Negotiation involving incompatible receivers.

bound as low as possible, that is to its initial value (because a decrease of the lower bound could violate the sender's requirements, which are based on the semantics of the data). The problem, illustrated in figure 5, appears when the intervals selected by the receivers are such that the strongest lower bound is greater than the weakest higher bound.

This corresponds to a situation where at least one receiver requires a minimum quality which is better than the best quality that can be offered to (or is requested by) at least one of the other receivers. Receivers with such conflicting values are said to be incompatible with each other. Because the requirements of incompatible receivers cannot be met at the same time, the only way out for the service provider is to select a subset of compatible receivers and complete the establishment of the communication with them only. The other (incompatible) receivers are notified of the impossibility to include them in the communication. We therefore see that the set of receivers that actually complete the communication set-up phase successfully may only be a subset of the set of users which had positively responded to the primitive indicating the establishment of the communication.

Of course, there exist several different subsets of compatible receivers. The service provider will finally pick up the subset of receivers that optimises utility functions as well as meeting some criteria, all of which reflect policies of the sender and/or the service provider itself. For example, the sender may impose a criterion called Active Group Integrity (AGI) [11]. AGI expresses conditions on the membership of the set of selected receivers, such as the identification of users that have to be part of the communication (key members), while the service provider may implement the policy of maximising the number of receivers. In

such a case, the selected set of compatible receivers would be the largest set including the key members. Of course, the service provider might not be able to form a set of compatible receivers meeting all the criteria. In such a case, the establishment of the communication fails. In our example, it is the case if some of the key members are incompatible.

We now describe an efficient algorithm to find the different subsets of compatible receivers on which the different criteria and utility functions can then be tested.

6. Compatible Subsets Finding Algorithm

6.1. Foundations

Set theory tells us that if a set consists of n elements, then the total number of its subsets equals 2^n. Fortunately, in our QoS negotiation, from all these subsets, only those containing compatible receivers will be of interest to us. A fundamental theorem, called the *compatible subsets theorem*, further reduces the number of subsets we will have to consider during the QoS negotiation phase of the establishment of point-to-multipoint communications.

Indeed, the theorem (see appendix) states that a compatible subset of receivers whose consensus minimum value is min_{cons} only includes compatible subsets whose interval values (min_{sub}, max_{sub}) verify the relation:

$$min_{sub} \leq min_{cons} \leq max_{sub} \tag{1}$$

One of the goals of the service provider should be to try to maximise the number of receivers taking part in a point-to-multipoint communication. Therefore, among all the subsets of compatible receivers exhibiting the same consensus minimal value, we will only be interested in the one with the greater cardinal (i.e. containing the greater number of receivers). Thanks to the previous theorem, we know that this subset is unique: indeed, if two such different subsets with the same cardinality existed, their union would still have the same consensus minimum value but a greater cardinality.

From this, we can conclude that for each different minimum value required by the receivers, we will only have to consider the biggest compatible subset of receivers yielding that value as the consensus minimum. The maximum number of different subsets that the service provider will have to consider has now been

reduced to $(initial\ max - initial\ min + 1)/granularity$. Here, initial min and initial max represent the interval proposed by the sender in the establishment request primitive and 'granularity' the minimum 'step' of variation that can be inflicted on the bounds of the interval.

In multimedia communications, the throughput granularity, for example, can be of the order of kbps or greater. This means that from an initial interval of 1 kbps to 10 kbps, with the throughput having a granularity of 1 kbps, a maximum of 10 subsets will have to be considered[7].

The reduction in the number of subsets to be considered, compared to the original 2^n, can be substantial, especially for large groups (i.e. large n).

6.2. Description of the Algorithm

From the previous section, it is clear that the goal of our algorithm is to find the biggest and unique subset of compatible receivers, for every different minimum requirement (expressed by these receivers). The principle followed by the algorithm derives from the observation that for a given minimum requirement, the final subset of receivers in which we are interested can be constructed by repeated application of the compatible subsets theorem.

The general working principle of the algorithm is then rather straightforward: the algorithm maintains a binary tree [13] whose nodes contain distinct minimum values (proposed by the receivers). The tree—the min-tree—is maintained so that at any node, the left sub-tree contains only nodes whose minimum values are (strictly) smaller than the minimum at the node, and the right sub-tree contains only minimum values that are (strictly) greater. Each node also contains the root of another binary tree—a max-tree, maintained in a similar way, but whose nodes contain the maximum values of the intervals whose minimum value is the one held in the corresponding node of the min-tree. The nodes of a max-tree also hold either a count of the number of receivers with the corresponding requirements, or a list of these receivers or both. Although the overall data structure is quite complex, it can be decomposed into several binary trees when considering minimum or maximum values in isolation. In a sense, the min-tree can be considered as "connecting" a forest of max-trees.

The subset of compatible receivers exhibiting a given consensus minimum value can therefore be computed quickly and easily by only visiting the nodes

[7] We also say that there are 10 *levels of minimum requirements.*

of the min-tree and corresponding max-trees whose values verify relation (1). Finally, because a consensus minimum value is necessarily in a node of the min-tree, all the subsets of interest can be computed by traversing the min-tree and considering, in turn, the values in its nodes as consensus minima. Figure 6 gives a pseudo-code description of the algorithm. The insertion of a receiver's requirements in the min-tree and corresponding max-tree is performed 'on the fly' (e.g. as the response arrives), which reduces the overhead introduced by the algorithm on communication establishment time (because responses are inserted during inter-response intervals). The insertion technique we used produces randomised binary trees, which improves the efficiency of the algorithm by keeping the trees balanced. This technique is outside the scope of this paper. For details, see, for instance, [13].

For each subset[8] of receivers, besides the consensus interval values, additional information can be kept in order to allow, or at least ease, the assessment of the criteria and the computation of the utility functions. Among others, the number of receivers in the subset (the subset's cardinality) will probably be of interest.

In environments where the underlying communication sub-system is receiver-oriented, that is where the receivers themselves join any underlying group communication, the final consensus interval is sufficient for each receiver to determine its membership to the selected subset (thanks to the "only if" part of the compatible subsets theorem). On the other hand, there are environments, said to be sender-oriented, where the inclusion of any receiver in an underlying communication can only be performed at the sender's request. In such an environment, the membership of every possible subset has to be explicitly archived (by the sender). Of course, such membership data consume computational power (which implies a degradation of performance) and memory space (which turns out to be a major handicap in the case of large groups of receivers). We therefore see that in the case of sender-oriented environments, which have long been known to be unsuited to large groups of receivers, our algorithm is just another penalty.

By combining the assessment of some criteria with the computation of the different subsets of receivers, an increase of performance can be achieved. For example, in the case of an AGI condition (see section 5.3) specifying key members,

[8] Note that the sender is a member of every subset by default. This is due to the rules of negotiation as explained in section 5.1, as well as to the compatible subsets theorem.

```
Compute_subsets (min-tree_root);
definition of Compute_subsets (min_tree: tree_node)
      if (tree_node is external) → return; endif
      Compute_subsets (right subtree);
      Find_in_min_tree (min-tree_root, value of tree_node);
      Test selection criteria and utility functions on this subset;
      Compute_subsets (left subtree);
end of definition
definition of Find_in_min_tree (min_tree: tree_node, cons_min_value)
      if (tree_node is external) → return; endif
      Find_in_min_tree (left subtree, cons_min_value);
      if (value of tree_node ≤ cons_min_value) →
            Find_in_max_tree (max-tree of this node, cons_min_value);
            if (value of tree_node < cons_min_value) →
                  Find_in_min_tree (right subtree, cons_min_value);
            endif
      endif
end of definition
definition of Find_in_max_tree (max_tree: tree_node, cons_min_value)
      if (tree_node is external) → return; endif
      Find_in_max_tree (right subtree, cons_min_value);
      if (value of tree_node ≥ cons_min_value) →
            Account for corresponding receivers in the subset;
            if (value of tree_node > cons_min_value) →
                  Find_in_max_tree (left subtree, cons_min_value);
            endif
      endif
end of definition
```

Figure 6. Pseudo-code for the algorithm.

whenever the interval requirement of one of these key members is being treated, the subsets in which the key receiver is not included can be discarded.

Finally, this basic method can easily be extended to provide additional functionality, such as finding a set of communications whose selected compatible subsets of receivers form a partition of the original set of receivers. This would yield

the optimisations described in section 4.

6.3. Complexity of the Algorithm

6.3.1. Space Requirements

Suppose that the initial interval proposed by the sender has a width of K levels of minimum requirements. The maximum size of the min-tree will thus be K nodes.

For a given node of the min-tree, we know that the maximum size of the associated max-tree will be equal to the difference of the initial max and the minimum value held by the node, augmented by one. Consequently, the maximum number of nodes we can find in the forest of max-trees is equal to $(1 + 2 + \cdots + K) = K(K+1)/2$.

Therefore, the absolute maximum of the number of nodes constituting the trees used by the algorithm is $K + K(K+1)/2 = K(K+3)/2$. This absolute bound on the space requirement of the algorithm only depends on the initial interval value proposed and is thus always known by the sender.

Furthermore, because there are at most one min-tree node and one max-tree node per receiver, we conclude that for a given negotiation, the maximum number of nodes required is:

$$M = min(K(K+3)/2, 2N) \tag{2}$$

where N is the number of receivers.

6.3.2. Time Requirements

From the theory of algorithm analysis, we know that it takes about $2 \ln n$ operations to search (and insert, if properly coded) in a randomised binary tree comprising n items [13].

Again, suppose that the width of the original interval proposed by the sender is K levels.

It takes about $2 \ln K$ operations to find a node in the min-tree (if the min-tree is full). Now, suppose that this node corresponds to the ith level of the negotiation interval. Thus, we know that the associated max-tree can only contain at most $(K - i + 1)$ nodes.

We assume that the selected values of intervals follow a uniform distribution: this is a worst-case assumption because, since each value is then equally likely to

be chosen, the corresponding nodes are all equally likely to appear and, therefore, the trees are likely to be full. Thus, on the average, to insert a receiver in a max-tree, it takes:

$$\frac{2}{N}\sum_{i=1}^{K}\ln(K-i+1)=\frac{2}{N}\ln K! \tag{3}$$

We conclude that, per receiver, the number of operations for an insertion is at most around:

$$2\ln K+(2/K)\ln K!\leq 4\ln K \tag{4}$$

This result justifies the algorithm used: insertion (i.e. installation of responses from the receivers in memory) is fast and scales well. Indeed, for large groups, $K \ll N$, the total insertion time is smaller than $4N\ln K$, that is, the total insertion time is linear (since K is a constant). Furthermore, let T be the lapse of time necessary for the algorithm to receive all the responses. As long as $4N\ln K \leq T$, the insertions take place during a period of time which would have otherwise been spent waiting for responses and therefore incur no extra time overhead. We call this condition the *free insertion time condition*.

To compute a subgroup of compatible receivers, say the one with minimum bound equal to the i^{th} level of the negotiation interval, the algorithm must:

1. find the first node of the min-tree whose minimum value is smaller than or equal to the i^{th} level of the interval, which takes no longer than $2\ln K$;
2. visit the i nodes of the min-tree whose values are smaller than or equal to the i^{th} level of the interval;
3. for each of the min-tree node, find the first node in the corresponding max-tree whose value is greater than or equal to the i^{th} level of the interval, which takes no longer than $2\ln(K-i+1)$;
4. visit the max-tree nodes whose values are greater than or equal to the i^{th} level of the interval. There are $K-i+1$ such nodes.

Therefore, to compute all the subgroups of interest, the number of operations required is at most:

$$\sum_{i=1}^{K}(2\ln K+2\ln(K-i+1)+i(K-i+1))=2K\ln K+2\ln K!+K(K+1)(K+2)/6 \tag{5}$$

Although this computation is cubic in K, we argue that since K represents the number of different minimum quality levels available, it will, in practice, be kept rather low (order of hundreds or less). Furthermore, it is worth noting that the computation of compatible subgroups is independent of N, the size of the group of receivers.

Because the computation of subgroups is the net time overhead introduced by the QoS negotiation if the free insertion time condition holds, and because it is independent of N, we conclude that the algorithm scales well with respect to the size of the group of receivers.

7. Considerations about Signalling

In section 5, we saw that the communication sub-system should be able to express its capabilities during the first phase of the end-to-end QoS negotiation (that is before issuing any establishment indication). Ideally, this would require network relay entities to be able to express their capabilities when forwarding the set-up request of the end-to-end communication. This suggests that the network should be able to see the packets it forwards as being related to each other (there should be a set-up phase within the network). From all the signalling protocols we have considered (i.e. UNI [2], DSM-CC [3] and RSVP [18,4]) only RSVP supports such a feature through the *One Path with Advertising (OPWA)* mechanism [14]. However, it should be noted that because OPWA simply advertises the resources without "booking" them, the information it provides may quickly become obsolete. This means that with current network signalling, transport protocol entities have either to "guess" the network capabilities or totally ignore the network during the end-to-end QoS negotiation.

The former case simulates an active role of the network during the QoS negotiation by, for example, a series of trials and failures on network communication establishment. Although this could provide semantics for the end-to-end negotiation which are close to those described in this paper, the efficiency of the negotiation could be dramatically reduced (i.e. the set-up time of the end-to-end communication could be increased). In the latter case, once the consensus values on the end-to-end communication are known, an attempt to establish a network communication among the end-systems hosting the subgroup of compatible receivers is made. Because of a lack of network resources, some of these receivers might be dropped *a posteriori*. Although such *a posteriori* rejections may doom

the creation of the end-to-end communication (because of a possible violation of an AGI condition, for instance), it would not be uncommon (incompatible receivers are also rejected).

It is also worth noting that none of the current signalling protocols, as well as none of the networks, distinguishes between the two classes of QoS parameters that exist in group communications. This leads to networks that are either too restrictive (imposing a consensus semantic to every parameter) or that use their resources sub-optimally (by violating the consensus nature of some of the parameters).

In designing network signalling protocols that are able to play an efficient and correct role in flexible end-to-end group communications, the idea of open signalling [9], where each network relay could play an active role, seems promising and is being investigated in our current work.

Also, interactions of such signalling protocols with QoS-routing methods (e.g. [16]) can play an important role in QoS negotiation.

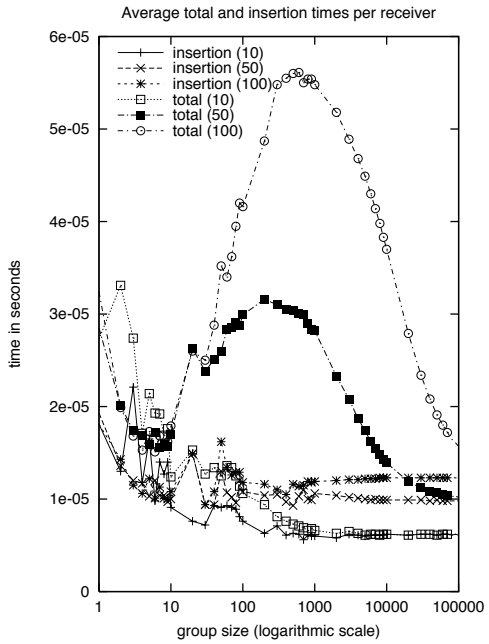
Finally, because of the results presented in section 6.3.2, it should be clear that network signalling is likely to account for a major portion of the overall set-up cost (i.e. time) of end-to-end group communications.

8. Simulations

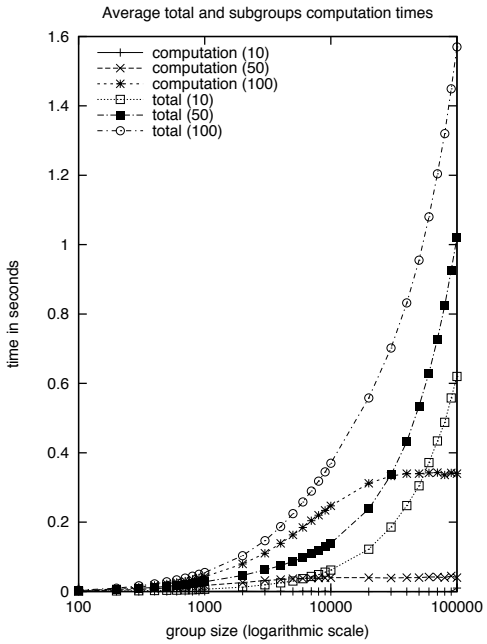
We tested our algorithm[9] on a PC equipped with a Pentium 166 MHz processor, 32 MB of RAM and running Windows 95. Our measurements were performed using the system's high-resolution timer whose frequency is about 1.2 MHz.

We ran the algorithm for various group sizes ranging from 1 to 100,000 receivers, each of them being tested with three QoS interval sizes (10, 50 and 100 levels respectively). The responses of the receivers were random, which is a worst case scenario, because it tends to "fill" the trees, or at least generate a bigger number of nodes than in the case where some patterns exist in the responses. Furthermore, the tested version is a "sender-oriented" one, that is, the algorithm keeps track of the membership of the subgroups, which also adds to both the spatial and time overhead of the algorithm. Also, several programming optimisations (e.g. memory pre-allocation) were deliberately not used in order to produce

[9] The results in this section are related to the performance of our algorithm only. That is, signalling overheads in the network are not taken into account (for reasons explained in section 7).



7.(a) Per-receiver average total (insertion and computation) and per-receiver average insertion times.



7.(b) Absolute average total and absolute average computation times.

Figure 7. Simulation results. The number of levels in the QoS intervals (10, 50 or 100) is indicated in parentheses. The scale of the abscissa axis is logarithmic.

real worst case values. For our simulations, the only utility function to be optimised was the number of receivers, with no other criterion (specifically, no AGI). Finally, each test was performed 500 times for statistical reasons. Figure 7(a) shows both the average total time and the average insertion times measured per receiver. Despite some fluctuations for small groups, the average per-user insertion time clearly exhibits a stable, constant value when measured over larger groups. Such a constant, whose value only depends on the size of the negotiated QoS interval, is in agreement with relation (4) (section 6.3.2) and confirms that the insertion time scales well.

The fluctuations of the per-user insertion time measured over smaller groups may seem at odds with relation (4). However, these are mainly due to task scheduling in the operating system: a small variation in the number of times a short task is suspended, as well as a small variation in the duration of the suspension, can have a dramatic influence on the total task time (as observed by

a user). For longer tasks such variations are less significant. Another reason that accounts for such fluctuations is that whenever a new tree node needs be created, a dynamic memory allocation is performed. For this reason, inserting a receiver that triggers the creation of a tree node is in practice more costly than inserting a receiver whose corresponding tree node(s) already exist(s).

For small groups, where tree nodes may potentially be created for each receiver, such an implementation overhead is responsible for the fluctuations observed. For larger groups, when the trees tend to become full, fewer tree nodes are created as the algorithm proceeds to treat the responses. Consequently, as the number of receivers increases, the cost of tree nodes creation is amortised, leading to more stable values for larger groups (especially when $N \gg K$).

The total time per receiver has a general tendency first to increase, then decrease toward the insertion cost, as the size of the group of receivers grows. This is because, for small groups, each receiver has potentially different requirements from the others, and the total cost of the algorithm is dominated by the subgroup computation part of the algorithm (since, by relation (5), we know that such a computation is roughly cubic in terms of the number of such requirements). For larger groups, the cost of the sub-groups computation reaches a maximum value independent of the group size (relation (5)). The total cost per receiver then becomes solely influenced by the insertion time, as the sub-group computation is amortised over more receivers. In other words, the total cost per receiver becomes independent of the group size, as this latter grows. Figure 7(b) shows that, in accordance with relation (5), the total sub-group computation time reaches a maximum independent of the group size. It also shows that for large groups ($N \gg K$), the total running time of the algorithm is a linear function of the group size (the apparently steep increase in the total time is due to the logarithmic scale of the abscissa axis). As explained earlier, this is because the total time is dominated by the insertion time.

This is an extremely important property: since insertions take place 'on the fly' (that is mostly during inter-response arrival periods), the major part of the time needed by our method, happens during time that would have otherwise been spent waiting. In other words, from a user point of view, our algorithm exhibits an overhead independent of the group size.

Experimentation therefore confirms that, in terms of group size, the method is optimal and scales well.

9. Conclusions

We have presented fundamental properties of QoS in group communications, leading to a classification of QoS parameters into two categories: *consensus parameters*, whose scope is the whole communication, and *receiver-dependent parameters*, whose scope is limited to each receiver separately. These properties (and consequently the classification) have been shown to be intrinsic.

The observation that, to the best of our knowledge, all the communication sub-systems available today ignore these properties, was the major motivation for this paper.

We have also shown how and why QoS negotiation is the central mechanism when tackling the heterogeneity found in group communications, especially in the transport sub-system (i.e. composed of the network and transport layers).

Then we have shown that QoS negotiation requires, in group communications, a three-way handshake, which, to be performed efficiently, requires support from network signalling that is, unfortunately, not yet available. We briefly suggested possible approaches to developing appropriate signalling mechanisms.

Finally, we have presented the design, analysis, and performance measurement results of a simple algorithm used in the negotiation of consensus QoS parameters. This algorithm proves to scale well in terms of the size of the group of receivers, and depending on future developments in network signalling, its net time overhead may turn out to be bounded by a constant that is independent of the group size. In other words, for large groups, we expect the signalling to be the potential bottleneck, not the negotiation.

This leads us to the conclusion that QoS negotiation within large groups is not only feasible, but can be done at low (time) cost. We argue that, in multimedia communications, an acceptable time set-up overhead can only be defined as a fraction of the total communication time, multiplied by a factor representing the size of the group of receivers. This is because on the one hand the set-up phase brings such advantages as performance guarantees, but on the other hand may be performed among quite large groups of receivers. Therefore, because end-to-end QoS negotiation adds only marginal overhead to the set-up time of group communications, this helps keep set-up overheads within the order of a few seconds (in the case of large groups), which is small compared to the lifetime of typical multimedia communications (typically minutes or more). We can thus foresee communications set-up techniques whose overhead is acceptable

to multimedia users.

References

[1] R. Ahuja, S. Keshav, H. Saran: Design, Implementation, and Performance Measurement of a Native-Mode ATM Transport Layer, IEEE/ACM Trans. Networking 4(1996) 502–515.

[2] ATM FORUM: ATM User-Network Interface (UNI) Signalling Specification—V4.0, June 1996.

[3] V. Balabanian, L. Casey, N. Greene: Digital Storage of Media-Command and Control Protocol Applied to ATM, IEEE J. Select. Areas Commun. 14(1996) 1162–1172.

[4] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin: Resource ReSerVation Protocol (RSVP)—Version 1 Functional Specification, Internet RFC 2205, September 1997.

[5] A. Campbell, G. Coulson, D. Hutchison: A Quality of Service Architecture, ACM SIGCOMM CCR 24(1994) 6–27.

[6] A. Danthine, O. Bonaventure, G. Leduc: The QoS Enhancements in OSI95, The OSI95 Transport Service with Multimedia Support, in: *Research Reports - ESPRIT*, A. Danthine (Ed.), Springer-Verlag, 1994, pp. 125–150.

[7] M. Dasen, G. Fankhauser, B. Plattner: An Error Tolerant, Scalable Video Stream Encoding and Compression for Mobile Computing, in: *Proc. ACTS Mobile Summit '96*, Granada, Spain, 1996.

[8] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C-G. Liu, L. Wei: An Architecture for Wide-Area Multicast Routing, ACM SIGCOMM CCR 24(1994) 126–135.

[9] A. Lazar, S. Bhonsle, K. Lim: A Binding Architecture for Multimedia Networks, J. Parallel and Distributed Comput. 30(1995) 204–216.

[10] L. Mathy, O. Bonaventure: QoS Negotiation for Multicast Communications, in: *Multimedia Transport and Teleservices (LNCS 882)*, D. Hutchison et al. (Eds.), Springer-Verlag, 1994, pp. 199–218.

[11] L. Mathy, G. Leduc, O. Bonaventure, A. Danthine: A Group Communication Framework, in: *Broadband Islands '94: Connecting the End-User*, O. Spaniol et al. (Eds.), Elsevier Science Publishers (North-Holland), 1994, pp. 167–178.

[12] H. Salama, D. Reeves, Y. Viniotis: Evaluation of Multicast Routing Algorithms for Real-Time Communication on High-Speed Networks, IEEE J. Select. Areas Commun. 15(1997) 332–345.

[13] R. Sedgewick: Algorithms in C, Parts 1–4, 3^d edition, ISBN 0-201-31452-5, Addison-Wesley, 1998.

[14] S. Shenker, L. Breslau: Two Issues in Reservation Establishment, ACM SIGCOMM CCR 25(1995) 14–26.

[15] R. Steinmetz: Data Compression in Multimedia Computing, Springer-Verlag Multimedia Systems 1(1994) 166–204.

[16] R. Vogel, R-G. Herrtwich, W. Kalfa, H. Wittig, L. Wolf: Qos-Based Routing of Multimedia Streams in Computer Networks, IEEE J. Select. Areas Commun. 14(1996) 1235–1244.

[17] N. Yeadon, F. Garcia, D. Hutchison, D. Shepherd: Filters: QoS Support Mechanisms for Multipeer Communications, IEEE J. Select. Areas Commun. 14(1996) 1245–1262.

[18] L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala: RSVP: A New Resource ReSer-Vation Protocol, IEEE Network 7(1993) 8–18.

Appendix

For each receiver R_i, we note the interval value representing the requirements by (min_i, max_i). The set containing all the receivers is denoted by $\mathcal{P}$.

For any subset $\mathcal{A}$ of $\mathcal{P}$ (i.e. $\mathcal{A} \subseteq \mathcal{P}$), we note the consensus values as $(min_{\mathcal{A}}, max_{\mathcal{A}})$, where $min_{\mathcal{A}}$ and $max_{\mathcal{A}}$ are defined as:

$$min_{\mathcal{A}} = max\{min_i \mid R_i \in \mathcal{A}\},$$
$$max_{\mathcal{A}} = min\{max_i \mid R_i \in \mathcal{A}\}.$$

Moreover, $\forall \mathcal{A} \subseteq \mathcal{P}, \mathcal{A}$ is said to be compatible *iff* $min_{\mathcal{A}} \leq max_{\mathcal{A}}$.

Lemma. $\forall$ sets $\mathcal{A} \subseteq \mathcal{P}$ and $\mathcal{B} \subseteq \mathcal{P}$, such that $\mathcal{B}$ is compatible and $\mathcal{A} \subseteq \mathcal{B}$, then $\mathcal{A}$ is compatible with $min_{\mathcal{A}} \leq min_{\mathcal{B}} \leq max_{\mathcal{B}} \leq max_{\mathcal{A}}$.

Proof. $\mathcal{A} \subseteq \mathcal{B} \Rightarrow \mathcal{B} = (\mathcal{B} \cap \overline{\mathcal{A}}) \cup \mathcal{A}$, and we have

$$\begin{aligned} min_{\mathcal{B}} &= max\{min_i \mid R_i \in (\mathcal{B} \cap \overline{\mathcal{A}}) \cup \mathcal{A}\} \\ &= max(max\{min_k \mid R_k \in \mathcal{A}\}, max\{min_j \mid R_j \in \mathcal{B} \cap \overline{\mathcal{A}}\}) \\ &= max(min_{\mathcal{A}}, min_{\mathcal{B} \cap \overline{\mathcal{A}}}) \geq min_{\mathcal{A}} \end{aligned}$$

Also,

$$\begin{aligned} max_{\mathcal{B}} &= min\{max_i \mid R_i \in (\mathcal{B} \cap \overline{\mathcal{A}}) \cup \mathcal{A}\} \\ &= min(max_{\mathcal{A}}, max_{\mathcal{B} \cap \overline{\mathcal{A}}}) \leq max_{\mathcal{A}} \end{aligned}$$

Because $min_{\mathcal{B}} \leq max_{\mathcal{B}}$, we conclude that $min_{\mathcal{A}} \leq min_{\mathcal{B}} \leq max_{\mathcal{B}} \leq max_{\mathcal{A}}$, and $\mathcal{A}$ is thus compatible. □

Theorem (Compatible Subsets Theorem).
In $\mathcal{P}$, $\mathcal{C} = \mathcal{A} \cup \mathcal{B}$ is compatible if and only if $\mathcal{A}$ and $\mathcal{B}$ are compatible, with $min_{\mathcal{B}} \leq min_{\mathcal{A}} \leq max_{\mathcal{B}}$ or $min_{\mathcal{A}} \leq min_{\mathcal{B}} \leq max_{\mathcal{A}}$

Proof. For the "if part", we have

$$min_{\mathcal{C}} = max\{min_i \mid R_i \in \mathcal{A} \cup \mathcal{B}\} = max(min_{\mathcal{A}}, min_{\mathcal{B}})$$

$$= \begin{cases} min_{\mathcal{A}} & \text{if } min_{\mathcal{B}} \leq min_{\mathcal{A}} \\ min_{\mathcal{B}} & \text{if } min_{\mathcal{A}} \leq min_{\mathcal{B}} \end{cases}$$

$$max_{\mathcal{C}} = min\{max_i \mid R_i \in \mathcal{A} \cup \mathcal{B}\} = min(max_{\mathcal{A}}, max_{\mathcal{B}})$$
$$= \begin{cases} max_{\mathcal{A}} & \text{if } max_{\mathcal{A}} \leq max_{\mathcal{B}} \\ max_{\mathcal{B}} & \text{if } max_{\mathcal{B}} \leq max_{\mathcal{A}} \end{cases}$$

In the case where $min_{\mathcal{B}} \leq min_{\mathcal{A}} \leq max_{\mathcal{B}}$, knowing that $min_{\mathcal{A}} \leq max_{\mathcal{A}}$, we can conclude that $min_{\mathcal{C}} = min_{\mathcal{A}} \leq min(max_{\mathcal{A}}, max_{\mathcal{B}}) = max_{\mathcal{C}}$. $\mathcal{C}$ is thus compatible. We come to the same conclusion when $min_{\mathcal{A}} \leq min_{\mathcal{B}} \leq max_{\mathcal{A}}$, by inverting the roles of $\mathcal{A}$ and $\mathcal{B}$.

Conversely, because $\mathcal{A} \subseteq \mathcal{A} \cup \mathcal{B} = \mathcal{C}$ and $\mathcal{B} \subseteq \mathcal{A} \cup \mathcal{B} = \mathcal{C}$ and $\mathcal{C}$ is compatible, by the lemma we have $\mathcal{A}$ and $\mathcal{B}$ are both compatible and $min_{\mathcal{A}} \leq min_{\mathcal{C}} \leq max_{\mathcal{C}} \leq max_{\mathcal{A}}$, and $min_{\mathcal{B}} \leq min_{\mathcal{C}} \leq max_{\mathcal{C}} \leq max_{\mathcal{B}}$.

Therefore, we have

$$min(min_{\mathcal{A}}, min_{\mathcal{B}}) \leq max(min_{\mathcal{A}}, min_{\mathcal{B}}) \leq min_{\mathcal{C}} \leq$$
$$max_{\mathcal{C}} \leq min(max_{\mathcal{A}}, max_{\mathcal{B}}) \leq max(max_{\mathcal{A}}, max_{\mathcal{B}})$$

and we can conclude that $min_{\mathcal{B}} \leq min_{\mathcal{A}} \leq max_{\mathcal{B}}$ or $min_{\mathcal{A}} \leq min_{\mathcal{B}} \leq max_{\mathcal{A}}$. □

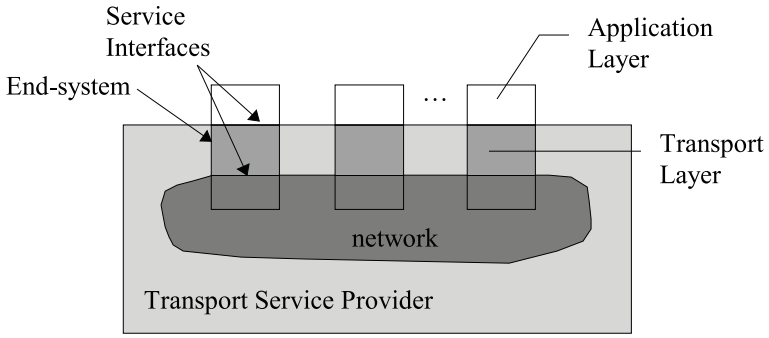


Figure 1. Simplified communication architecture, showing three end-systems connected to the network.

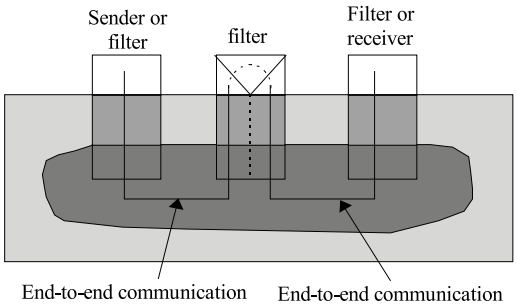


Figure 2. Filters as application gateways.

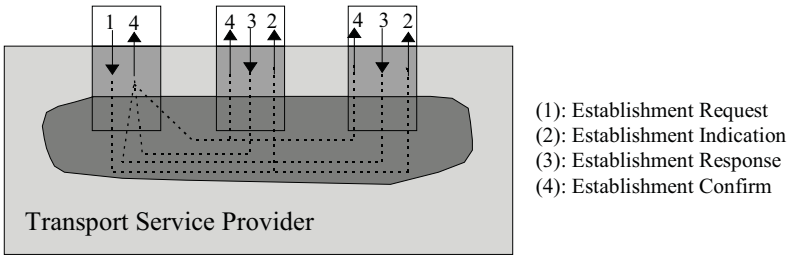


Figure 3. Three-way handshake.

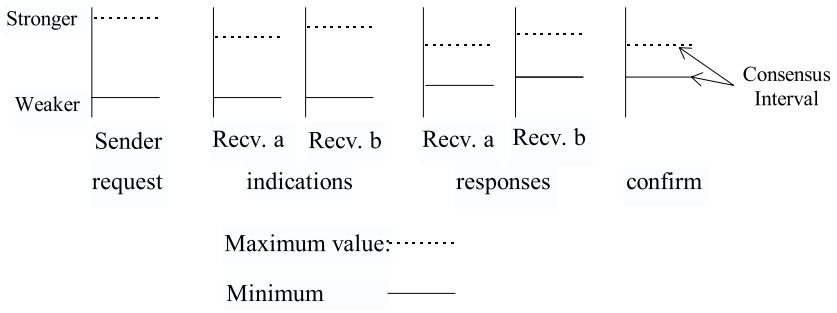


Figure 4. Rules for interval values negotiation.

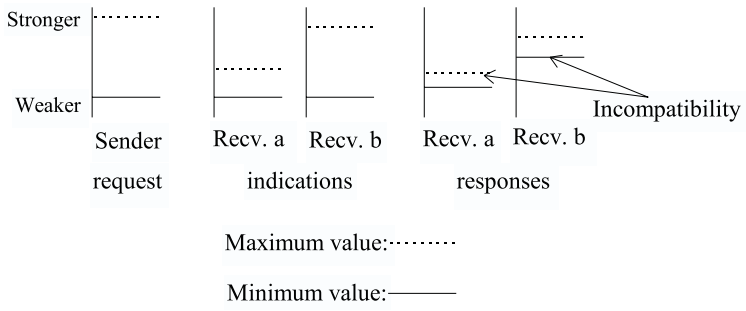
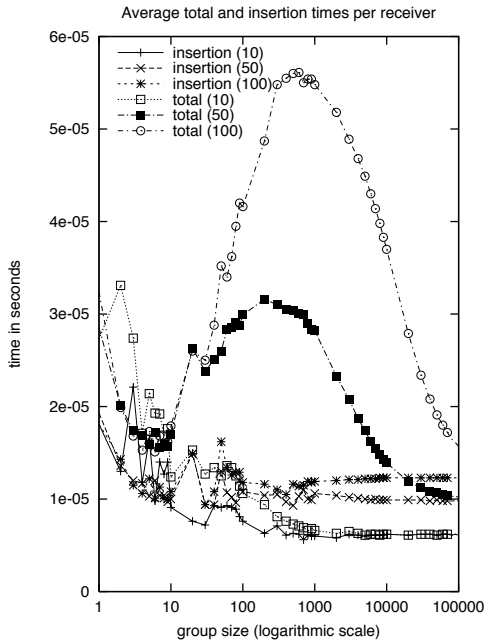


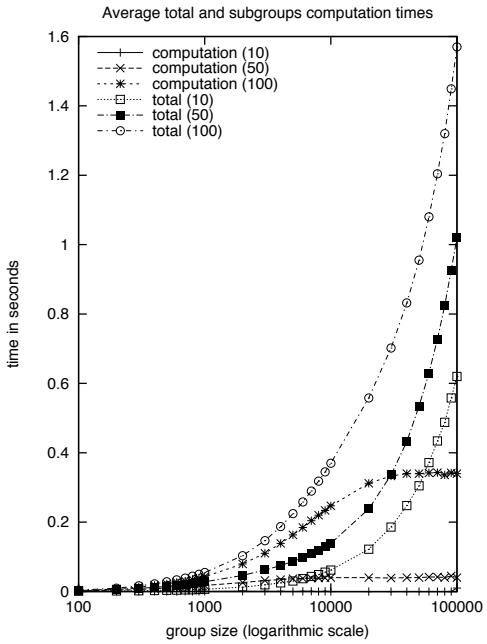
Figure 5. Negotiation involving incompatible receivers.

```
Compute_subsets (min-tree_root);
definition of Compute_subsets (min_tree: tree_node)
      if (tree_node is external) → return; endif
      Compute_subsets (right subtree);
      Find_in_min_tree (min-tree_root, value of tree_node);
      Test selection criteria and utility functions on this subset;
      Compute_subsets (left subtree);
end of definition
definition of Find_in_min_tree (min_tree: tree_node, cons_min_value)
      if (tree_node is external) → return; endif
      Find_in_min_tree (left subtree, cons_min_value);
      if (value of tree_node ≤ cons_min_value) →
            Find_in_max_tree (max-tree of this node, cons_min_value);
            if (value of tree_node < cons_min_value) →
                  Find_in_min_tree (right subtree, cons_min_value);
            endif
      endif
end of definition
definition of Find_in_max_tree (max_tree: tree_node, cons_min_value)
      if (tree_node is external) → return; endif
      Find_in_max_tree (right subtree, cons_min_value);
      if (value of tree_node ≥ cons_min_value) →
            Account for corresponding receivers in the subset;
            if (value of tree_node > cons_min_value) →
                  Find_in_max_tree (left subtree, cons_min_value);
            endif
      endif
end of definition
```

Figure 6. Pseudo-code for the algorithm.



7.(a) Per-receiver average total (insertion and computation) and per-receiver average insertion times.



7.(b) Absolute average total and absolute average computation times.

Figure 7. Simulation results. The number of levels in the QoS intervals (10, 50 or 100) is indicated in parentheses. The scale of the abscissa axis is logarithmic.