[GLE+01] *Carsten Griwodz, Michael Liepert, Abdulmotaleb El Saddik, Giwon On, Michael Zink,Ralf Steinmetz;* **Perceived Consistency;** ACS/IEEE International Conference on Computer Systems and Applications, Beirut,June 2001. S.

Perceived Consistency

Carsten Griwodz[2], Michael Liepert[1], Abdulmotaleb El Saddik[1], Giwon On[1], Michael Zink[1] and Ralf Steinmetz[1]

1
Industrial Process and System Communications
Dept. of Electrical Eng. & Information Technology
Darmstadt University of Technology
Merckstr. 25 • D-64283 Darmstadt • Germany

2
University of Oslo
Department of Informatics
Postbox 1080 Blindern N-0316
Oslo Norway

{griff, lipi, abed, giwon, zink}@KOM.tu-darmstadt.de

Abstract

Quality of service guarantees for multimedia communication systems have been considered on several abstraction levels. In the multimedia networking field it is typical to identify the minimal QoS requirements of an application to save resources by guaranteeing its functionality. Many of these applications can operate in spite of an imperfect delivery of media data, while other applications such as distributed databases or distributed filesystems consider perfect QoS necessary but accept delay. The basic problems of the latter is the consistency of their data, while the former require a consistent perception of the content. More generically, both QoS requirements can be interpreted as a problem of maintaining a consistent system state. Consequently we assume that many distributed applications, including most distributed multimedia applications, can fulfil their tasks in spite of imperfect consistency. Since the application requirements differ widely, the elements that make up "consistency" must be separated and classified. This paper introduces Consistency QoS and proposes a classification of elements that determine an application's consistency requirements. The low level QoS requirements that these separate parameters rely on are shown, and example parameter sets for application classes are given.

1. Introduction

The number of multimedia communication systems has increased rapidly in recent years. These systems have to handle data types that are different from traditional data types like text or HTML files and therefore require a different behaviour of the network and the applications. Limited bandwidth, for example, is annoying but rarely critical for file transfer. However real-time playback of video and audio streams becomes usually impossible if it falls short of a lower bandwidth limit. In order to solve this problem, quality of service (QoS) mechanisms for multimedia communication systems have been considered on several abstraction levels. The use of these mechanisms allows distributed applications to handle multimedia contents.

In the beginning, the focus of QoS-related research in multimedia communication systems was on network level QoS. Its goal is to identify and negotiate the minimal QoS requirements of an application instead of over-provisioning the network resources. Since negotiations do not always result in guaranteeing the best service to the application, two different application categories can be identified:

Applications that can operate in spite of an imperfect delivery of data. A limited number of errors in such data does not harm the application but may reduce the quality that is delivered to the user.

Applications that do not allow any errors in the transmitted data but accept other performance disadvantages including unpredictable delay and jitter

The basic requirement for the second category is the consistency of their data, while the first category requires a consistent perception of the data. More generically, both kinds of application require maintenance of a consistent system state. On the one hand, the focus is mostly on an error free transmission of the data, on the other hand it is on a small or no delay. In both cases this consistency is increasingly hard to achieve when the scale of a distributed system grows.

In this paper we express the hypothesis that several kinds of distributed applications do not fall into the two clearly cut categories. We assume that many distributed applications, including most distributed multimedia applications, can fulfil their tasks in spite of imperfect consistency. Section 2. will motivate this hypothesis with some specific applications that are already in use.

We realize that the requirements of distinct applications differ widely and therefore, that the elements which make up "consistency" must be separated and classified. This separation and classification will make it easier to exploit the limits to a distributed application's consistency requirements in generic ways and potentially, to increase its scalability. Once a classification exist, it will become easier to specify conditions of inconsistency that still allow correct operation of an application. The availability of a classification can also be taken into account during the development of an application in order to gain scalability

We propose a classification of elements that will allow the specification of an application's consistency requirements and thus define the maximum state of

imperfectness that still allows an application to work correctly. We show low level QoS requirements that each parameter relies on.

2. Related Work

After the introduction of lower layer QoS support like the Tenet group's [6] higher level QoS architectures appeared soon. XRM [9] was one of the first that implemented a generic QoS architecture The highest practically relevant abstraction level to date is the integration of QoS handling into CORBA, which was investigated by [3], [11] and [14]. Many other QoS enhanced applications like [2] and [16] have followed the approach of direct mappings and simplifications without any generalized abstraction. In other application areas, the term QoS is not applied, although service guarantees are their central concern.

To have a limited set of examples available later in the text, we start with five real-world examples of distributed applications that have a distributed system state but do not rely entirely on a hard synchronity between the copies of the data. These examples can not cover the entire range of aspects that need investigation, but our initial investigations are limited to the covered set of aspects.

2.1 Computer Games

A distributed, interactive computer game has a single environment that is concurrently manipulated by all participants. However, there are rules. Some things need not be kept consistent because they do never change (Walls in MiMaze [7], the race track in a car racing game [12]). Some things can be changed by only one present person, not by others (target car speed). Some things are always accepted (a turn of the steering wheel is always accepted, since user input is never rewindable). [12] has demonstrated that a highest permissible delay in system feedback exists for applications such as the his racing game, and that processing variations can reduce the effects of such delays on the user perception. Investigations showed that with a delay of 50 ms a player is still able to steer the car without a perceptible reduction of his control.

2.2 Vehicle Remote Control

In this example we assume the rather extreme situation of a single vehicle like the Pathfinder for the Mars mission or other remotely controlled vehicles that are needed to fulfil special tasks. This application differs from e.g. the games scenario in several ways. An important one is that the possible activity of the environment is not limited to preprogrammed actions. The system can be consistent for a very long time, but it can suddenly become inconsistent by an unforseeable event. In normal operation, for the Pathfinder specifically it was predictable that there were no other remotely controlled vehicles or moving objects, thus future situations could be predicted on the basis of the vehicle's movement alone. As this example shows, it can be valuable for distributed applications to consider which data in the physical environment are subject to change and which are not.

2.3 Distributed multimedia filesystems with disconnected operations

A distributed file system like the CODA file system [10] will usually refer to a reference copy when it operates in connected mode, not allowing the system to become inconsistent. In disconnected mode, the occurrence of inconsistencies is expected, and notifications and user-controlled mechanisms are provided to resolve such inconsistencies.

This type of filesysytem supports disconnected operation mainly for mobile clients. Unlike other distributed file systems like DFS [5] and AFS [4], CODA supports the continued operation during partial network failures as well. This feature automatically introduces a certain level of inconsistency in distributed file systems. Update speed and frequency are small in this system since changes to the stored data would not occur in intervals of milliseconds Versioning of the files allows to rearrange consistency when a mobile system that has been disconnected reconnects.

2.4 Lip Synchronization

Loosing lip synchronization is a problem that appears in the playback of separately transported audio and video streams that belong together. This problem can be interpreted as an issue of inconsistent clocks. Although several synchronization mechanisms exist to eliminate this problem altogether, it may be possible to save resources by synchronizing such streams sufficiently well rather than perfectly. The definition of sufficiency is difficult, but Steinmetz [15] showed that the skew between audio and video belonging to the same presentation can be up to 80 milliseconds without being noticed by the casual viewer.

2.5 Mobile IP

Mobile IP addresses issues of lossy connections and, in general, unreliable and dynamic networks [13]. Services, applications and work practices that were designed for stationary, non-moving users linked to fixed networks have to be adapted to accomodate requirements imposed, constraints introduced, and possibilities opened by the mobility of users. Mobile IP extends and introduces protocols to allow for e.g. messaging, localization, security in such dynamic environments.

3. Introducing Consistency QoS

During our work on distributed systems we learned that some of these can deal quite well with a certain lack of consistency. Dealing with the problem of distributing changes of the system state sufficiently fast between the separate nodes in a distributed system, we call this flavor of QoS "Consistency QoS". It expresses all QoS issues in terms of constrained errors in a distributed system state. The examples of Section 2. indicate that many problems can be formulated in these terms. It is important to note that we do not address the problems of real-time applications alone, but try to guarantee QoS for applications that are not considered real-time as well. In a middleware architecture that is based on the replication of state, we assume that the various copies of state (data) are not perfectly consistent at all times. Next, we introduce the required concepts of *perceived consistency* (Section 3.1) and *perception* (Section 3.2). Based on this, we define the term *Consistency QoS* in Section 3.3, and then we present the list of Consistency QoS parameters that we have identified so far (Section 3.4).

3.1 Perceived Consistency

We assume that QoS requirements are often both application-dependent and exist inherently within applications. Finally, these requirements are driven by the conscious perception required of the user [8], respectively the required physical output. The examples in Section 2.1 and Section 2.4 provide an intuition of the term 'perceived consistency': if the end user believes that the video and audio streams are lip-synchronous, or if the user believes that the feedback to his controls are instant, perceived consistency is achieved. We define

Perceived Consistency for distributed systems means that the perceived physical output is interpreted equally by all users

An application can support perceived consistency only, but obviously never guarantee correct interpretation by the users. Still, support of perceived consistency puts demands onto applications.

For most areas that require consistent system states, the user involvement is less direct than in the given cases of Section 2. The implementation examples present special investigations or developments that are concerned with various scales of modifications to a system state. They seem to have few things in common. However, when the elements of a consistent system state are separated into measurable aspects, we can identify for each of the indicated applications a set of consistency requirements.

Starting with a reduced application space, we consider only distributed applications with a logical system model that can be reduced to a predictable and manageable set of atomic data. We simplify further by considering only simple, fix-sized data elements in a first step. Although we are not sure whether a generalization is possible starting with this approach, there are applications fulfilling these assumptions (Section 2.1 to Section 2.4), i.e. such applications that gain from an appropriate Consistency QoS infrastructure.

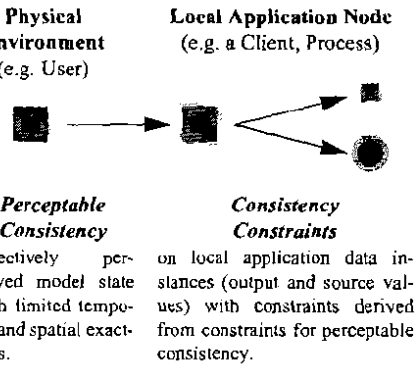

Figure 1: Perceptable Consistency and local Consistency Constraints

In a typical distributed application, some changes to the system state are not necessarily perceived immediately by each user. Actually, the same is true for some applications as well, so we can more generically talk about the physical environment of the distributed application. To reduce the effort for maintaining consistency, we try to find the implementation constraints at each edge to the physical environment, that still provide perceived consistency. To stay within these constraints, we map them onto constraints for their associated local instances of distributed application data (Figure 1). Having this, we want to benefit from loosened constraints on the respective application model values by cutting communication cost, preferably to the level that the physical and human environment of the application demands. This requires the notion of two concepts:

- perception of data at a node of a distributed application, presented in Section 3.2
- consistency constraints on perceived data, presented as Consistency QoS parameters in Section 3.4

3.2 Perception

Applications can exploit the fact that data need not be more up-to-date or exact than the perceived consistency. Of course, the permissible deviation from an actually consistent behaviour differs from one user to another, and an ideal perceived consistency can not be measured. Just as in lossy multimedia codecs, the permissible loss of perfect replication of the system state competes with the resources

that are required to achieve it, and the limits are chosen *subjectively*, e.g. *based on case studies*.

The concept of perception implies that availability of manipulated yet unperceived data is not relevant. The local node needs no consistent information about anything out of its current scope of operation. In other words the local node works consistently within its scope even if it does not have perfectly correct information about data outside its scope, granted that its copies are invalidated before they are perceived again.

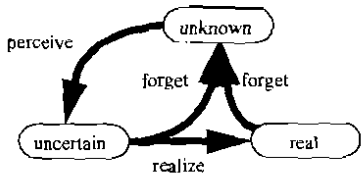

Figure 2: state graph for datum visibility

If data is manipulated by a user, or referred to for the first time, the result of the operation can not be known immediately Figure 2 shows the states for a datum at a local node that enters (*perceive*) respectively leaves (*forget*) the perception of the local copy of the application The state *real* indicates a reliable, non-rewindable value of the datum. This can obviously not be achieved immediately since the current state must be transferred from another node initially.

The perception state model is in so far oblivious of the communication delay between the nodes of the distributed system and problems of interactions with the physical environment. It only specifies, whether conflicts have to be resolved. Assume that a user A inputs a value for datum D, this datum is in the focus of user A and A's datum instance has therefore to be *real*. Any *uncertain* instance of D can be informed with a one-way message, nodes where D is unknown do not maintain instances of D. But if a user B at another node of the distributed application tries to manipulate datum D, the two respective instances of D are *real*. The expected results of these inputs may differ for users A and B, but the user input has been accepted by the respective nodes. Being physical input, the input is not rewindable and must be consumed, but the effect of the user inputs on D is not necessarily the respectively expected effect. Two situations can potentially occur:

- The conflict between the two nodes can be resolved within a delay acceptable for the distributed application. A model-dependent merging of the two user inputs is performed and the new state of datum D is consistently shown to users A and B.
- The communication delay between the two nodes is not acceptable. An application-defined exception handling must be performed.

3.3 Consistency QoS

Consistency QoS is intended to formalize the ability of an application to execute correctly in spite of data being inconsistent in some ways. With Consistency QoS, we try no more to maintain absolute consistency (i.e., the logical axioms of the respective application), but try to identify and maintain the perceptable consistency (i.e., perceptable aspects of these axioms). We define:

Consistency QoS is a contract between an edge of a distributed application and an underlying application layer that quantifies guaranteed contraints on request to and provision of change to a distributed system state

An edge of an application here is a layer that connects the application with the physical environment, e.g. GUIs or device adapters. The constraint quantification needs a scheme which is developed in Section 3.4. This scheme may be communicated to the physical environment including users in a potentially simplified presentation (like a slider to control smear effects).

Specifically, this scheme provides a means to specify acceptability for the conflict resolution mentioned for the perception state model in Section 3.2

3.4 Consistency QoS Parameters

We aim at the identification of a set of constraints suitable to describe the requirements which a perceived consistency may put on an application node. We try to formulate these constraints as application level QoS parameters, to be negotiated with a middleware.

Section 2. shows that datum consistency problems can not be specified by a single parameter. Similarly, it is relevant to understand whether intermediate steps of a series of consecutive state changes in an instance can be ignored when the state in another instance is updated. Consistency QoS can apply other QoS definitions as a basis for its guarantees (e.g. weakly consistent state can only be guaranteed when the end-to-end delay is known).

In fact the consistency problem can be split into several parameters and applications have different requirements in each of these parameters. Being an abstraction from lower system levels, the values for several of the parameters can only be achieved if the system is supported by QoS guarantees on lower levels: the enforcement of parameters for consistency must be supported by network level QoS and local system QoS (CPU, memory, disk access). This can be implemented by the use of e.g. an Integrated Services infrastructure [1] and an operating system and application able to provide QoS guarantees. Up to now we have identified the parameters of Consistency QoS that are presented in Table 1. In case of applications that rely on network level QoS guarantees, Consistency QoS can reduce the amount of resources that need to be reserved to guarantee the correct

parameter	distributed multimedia filesystems with disconnected operations (CODA-like)	vehicle remote control (Pathfinder)
update speed	slow	fast
update frequency	rare	frequent
synchronization frequency	days	seconds
replicability	high	limited
rewindability	manual	very limited
acceptability	none	~ update speed
local transitivity	versioned files	built-in (parameter interdependance is part of the system); very limited

Table 2. Consistency QoS parameters for extreme applications

variables of the supported data types can be used independently of their distribution state The data types are provided by the middleware: rather than using a standard integer data type *int* of the programming language, interfaces of data types such as *RewindableInteger* or *MergableInteger* with a limited set of operations are used at the application level. At the middleware level, each variable of a supported type is implemented with a specific conflict resolver. Three kinds of implementation are shown in Figure 3. The *simple data type implementation* has a locally available state, and all operations are timestamped and performed on the local copy as well as broadcast to all remote copies. Because of limits to the number of replica, a node in a distributed application may sometimes not hold a local instance of a specific datum. In that case, and if network resources are sufficient for that, a *remote simple data type stub* is made locally available It redirects operations to the remote instances and retrieves the state synchronously when it is requested by the application. References are a means for sharing dynamically created data among nodes. The *resolved reference implementation* allows data that is replicated to the local node without an interface at the application level, if they can be referred to by a reference Such a reference datum, which may be distributed itself, is used like a pointer by the application.

The raised grey box in Figure 3 provides a look into the main element of a local implementation, the conflict resolver. Conflict resolvers implement a specific strategy such a *rewinding* or *merging* to provide the guaranteed consistency. The box shows a conflict resolver that is based on a verified state that can not be manipulated any more. It maintains a queue of unverified operations that will be applied to the verified state when the time for a re-ordering insertion of operations from a remote node has expired. All retrieval operations of the application refer to a visible state that does not necessarily consider all operations that have already been performed on the datum instance; the visibility of the operations at the application level is *delayed*. Further important components of the middleware layer are the garbage collection for dynamical data and the namespace administration which is responsible for globally valid references.

We circumvent a central object naming service by identifying names with references When an instance first refers to a datum, a reference is created at the given node and its reference is registered. Since a reference itself is a datum, each datum is either connected to globally known data of the well-known binary code of the distributed application through a chain of references, or it is not. In the second case, this datum is an intermediate datum for the processing in a piece of code of one instance of the application. In the other case, this datum may become interesting to other instances of the application, but only if the datum is referred to. This reference by a remote instance can be achieved when the reference chain that allows location of the datum is distributed to the remote instance. When the remote instance de-references the datum that refers to the newly created datum, it acquires access to the original datum and perceives it. The means of such access are defined by the modes of the datum

The modes may forbid replication, which increases delay but guarantees consistency, or they may limit the overall number of replica in the distributed application. If replication is possible in spite of such conditions, requirements on the network level QoS must be checked to determine whether the temporal limits to the update speed of the datum's replica can be guaranteed. If this is impossible, the replication is not performed but each access to the datum by the remote node is executed by a remote call.

parameter name	meaning
update speed	The largest interval that is needed to deploy a change of state in a datum to all instances of that datum. Guarantees of update speed require guarantees on end-to-end delays at the network level.
update frequency	The highest frequency of state changes in a datum that can be handled by the service provider without affecting other guarantees. Guarantees on update frequency require guarantees on throughput and loss at the network level, and depend heavily on multicast features.
synchronization frequency	The lowest frequency of communication between any two instances of a datum that still allows to recreate a common synchronized state. Guarentees of synchronization frequency require network level guarantees on throughput, loss and delay.
replicability	The number and distribution of replica that may exist of a datum (replicability=1 makes most other Consistency QoS parameter irrelevant but may reduce system availability). Guarantees on replicability are based on application-defined constraints.
rewindability	The past states, in terms of granularity and past time, to which a datum can return. Guarantees on rewindability require local guarantees on available memory.
acceptability	The acceptable level of divergence of the current content on the display from the actual state of the system (e.g. due to human perception). Guarantees on acceptability are based on application-defined constraints.
local transitivity	The number of steps that can be reversed in modification of related local data when a datum is informed about a remote state change that occurred (legally w.r.t. update speed) before operations were performed based on incorrect information. Guarantees on local transitivity require local guarantees on available memory and computing speed

Table 1. Consistency QoS Parameters

functionality.

Without limits to the parameters of Table 1, an unacceptable number of rewind operations and re-calculations may become necessary. The resource usage of all approaches that allow re-integration of the system state is growing too quickly for most applications. Only if limitations apply, Consistency QoS is applicable at all. Table 2 shows that the requirement range of a single parameter can be very wide (see update speed or frequency).

We are investigating whether these QoS parameters are sufficient, independent, and atomic. With the given parameters, we can already demonstrate connections between low level QoS parameters and application-level parameters. We believe that the consideration of these parameters will in some cases allow applications to decide their QoS requirements without bothering with user interaction at all, in some cases allow the indication to a user that perceived quality will be severely disturbed and can not be increased, and finally provide users with means of setting QoS parameters in terms that are far more intuitive than, e.g. the packet loss ratio.

4. Implementations

The ongoing implementation is focused on early application. Thus, we implement a variety of consistency strategies (interpolation, voting, rewind, prediction, output delay) for some basic data types (numbers, boolean and character values), references and aggregations. As a proof of concept, we design and implement a simple middleware for a system that can provide Consistency QoS to applications on the basis of lower level QoS provision. Only after successful evaluation, it will be reasonable to integrate Consistency QoS into more complex and common frameworks, e.g. as a CORBA object adapter or CORBA 3.0 QoS policies.

We start by considering a distributed application that is implemented by separately running copies or entities. Being distributed, that application tries to maintain a consistent system state in spite of temporal delays in the distribution of changes to the system state from one node to another. As stated above, we have reduced the data types for initial examination. We do not consider more complex constructs such as classes and methods, we consider only precompiled, static application code at this time. It is currently uncertain in which way our results need to be adapted to apply to more dynamic setups.

Consequently, we consider a middleware that consists of an abstract distribution system that can guarantee network level QoS such as end-to-end delay and reliability, and that provides multicast facilities. Figure 3 shows a design of an infrastructure that considers simple data types and operations on these data types. At the application level,

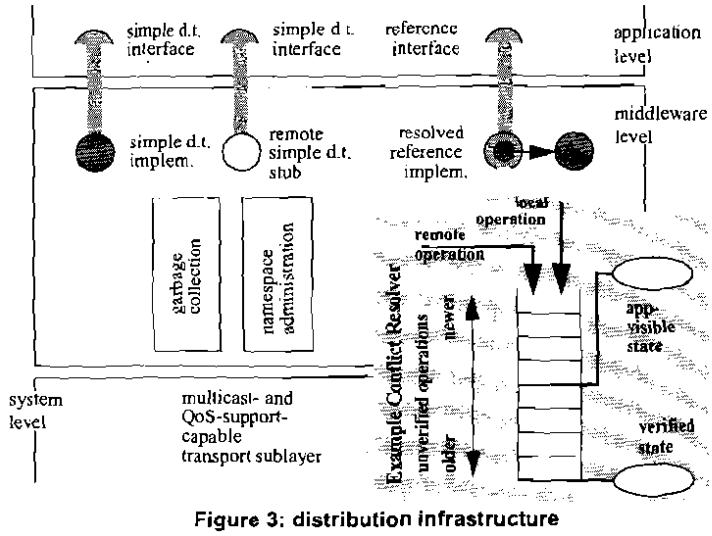

Figure 3: distribution infrastructure

5. Future Work

After the ongoing implementation of Consistency QoS, we will try to dissect applications and implant our shared partially consistent data types to get a first proof of concept and applicability.

Then, we want to implement an extensible framework to allow implementation of new "plug-in strategies" and also of new value types by means of aggregation (records, arrays and classes). Also, we want to have a look at the possibilities to integrate Consistency QoS implementation and concepts into CORBA.

Having a development framework at hand, the benefits of Consistency QoS have to be evaluated. This will certainly demand further refinement of Consistency QoS classification, application analysis and middleware implementation.

6. References

[1] Bob Braden, David Clark, and Scott Shenker. Integrated Services in the Internet Architecture. Internet RFC 1633, June 1994.

[2] T. Braun and S. Giorcelli. Quality of Service Support for IP Flows over ATM. In *Proc.s of Kommunikation in Verteilten Systemen:GI/ITK Fachtagung*. Springer-Verlag, Feb. 1997.

[3] Christian Becker and Kurt Geihs. Generic QoS Specifications for CORBA. In *Proceedings of KiVS'99, Kommunikation in Verteilten Systemen*, pages 184–195. Springer Verlag, March 1999.

[4] Richard Campbell. *Managing AFS - The Andrew File System*. Prentice-Hall, 1998.

[5] DFS Administration Guide. Transarc DCE Documentation, 1995.

[6] Domenico Ferrari, Anindo Banerjea, and Hui Zhang. Network Support for Multimedia. *Computer Networks and ISDN Systems*, 26(10), 1994.

[7] Laurent Gautier and Christophe Diot. Design and Evaluation of MiMaze, a Multi-player Game on the Internet. In *Proc. of IEEE Multimedia Systems Conference*, June 1998.

[8] William James. The Principles of Psychology - CHAPTER X: The Consciousness of Self. online library, 1890.

[9] Aurel Lazar, Shailendra Bhonsle, and Koon Seng Lim. Binding Architecture for Multimedia Networks In *Proc. of the International COST 237 Workshop*, pages 103–123. Springer-Verlag, Nov. 1994.

[10] Y.W. Lee, K.S. Leung, and M. Satyanarayanan. Operation-based Update Propagation in a Mobile File System. In *Proc. of the USENIX Annual Technical Conference*, June 1999.

[11] Klara Nahrstedt and Jonathan Smith. The QoS Broker. *IEEE Multimedia*, 2(1):53–67, May 1995.

[12] Lothar Pantel. Möglichkeiten zur Behandlung der Ende-zu-Ende Verzögerung in Mehrparteienspielen. Thesis, Jan. 2000.

[13] C. Perkins. RFC 2002 - IP Mobility Support. RFC, Oct. 1996.

[14] T. Plagemann, A. Saethre, and V. Goebel. Application Requirements and QoS Negotiation in Multimedia Systems. In *Proc. of Second Workshop on Protocols for Multimedia Systems*, October 1995.

[15] Ralf Steinmetz. Human Perception of Jitter and Media Synchronization. *IEEE J. Selected Areas on Communications*, 14(1):61–72, January 1996.

[16] Jens Schmitt, Michael Zink, Lars Wolf, and Ralf Steinmetz. Quality of Service Support for recording and playback of MBone session in heterogeneous IP/ATM networks. In *Proc. of SYBEN 98*, volume 3408, pages 374–383, May 1998.