QoS Monitoring in High Performance Environments

Claudia Schmidt, Roland Bless
Institute of Telematics, University of Karlsruhe
Zirkel 2, 76128 Karlsruhe, Germany
e-mail: [schmidt,bless]@telematik.informatik.uni-karlsruhe.de

Abstract

Forthcoming applications, especially distributed multimedia and interactive applications, have stringent requirements on the quality of the communication service. Therefore, an *integrated Quality-of-Service (QoS) management* is needed including all involved components to guarantee a communication service from application to application. One promising approach to guarantee a certain level of service uses the *reservation of communication resources* and the assignment of these resources to individual data streams based on the requested service. However, often it is not sufficient to reserve resources only. Additionally, *monitoring* of QoS parameters must be applied to ensure that the negotiated QoS is achieved. Within this paper, the design and implementation of a QoS monitor for communication subsystems is presented.

1. Introduction

Emerging communication subsystems need to integrate concurrent support for an increasing variety of applications, including audio, video, and conventional data traffic. Generally, such applications are characterized by highly different service requirements commonly expressed in terms of so-called *quality of service parameters* (QoS parameters). Most importantly, they require specialized end-to-end QoS support, i.e., from application to application. Thus, it is not sufficient to reserve network resources, but additionally resources must be reserved in all involved end systems [4]. Basic communication resources shared between several data streams are bandwidth of transmission links as well as CPU and buffer space in end and intermediate nodes.

From the resource reservation point of view, three *QoS building blocks* can be build around the communication resources, with each one responsible for several resources [13]. The identified building blocks are the *communication subsystem*, the *operating system* and the underlying *network*. The network covers medium and medium access, e.g., the MAC layer of a LAN or the ATM layer of an ATM-based network. Resource managers inside the network manage bandwidth as well as CPU and buffers in intermediate nodes. The second building block, the communication subsystem, includes all communication related functions that enhance the network service in order to provide services requested by applications. Resource managers have to configure several protocol functions to support service requirements. Finally, communication subsystem and application processes need operating system support in end systems. Resource managers of the operating system (e.g. scheduler or memory manager) handle resources, such as processor capacity and buffer space.

This paper discusses QoS management in the communication subsystem with special emphasis on QoS monitoring. Section 2. provides a short overview over QoS management functions. Following this, section 3. presents our model of the communication subsystem including all involved components. The reminder of this paper is concerned with monitoring aspects. In section 4. the design decisions and the resulting architecture of our QoS monitor are presented. Moreover first implementation efforts are described. Finally the paper is concluded by a summary and a description of future work.

2. Functions of an Integrated QoS Management

Building QoS oriented communication services deals with several QoS management functions. In detail, QoS management includes all functions that have to be performed to directly support application-tailored services [7]. QoS management should therefore co-ordinate the co-operation of all involved components automatically and transparently for the application. Application specific requirements are used to configure all mechanisms that are necessary to provide the application specific service. QoS management functions are:

- QoS specification
- QoS mapping

- QoS negotiation
- QoS maintenance and monitoring

QoS specification is concerned with the definition of the requested service. Several service interfaces on highly different abstraction levels are normally located inside of communication systems. Especially applications and system level components prefer to operate on highly different abstraction levels. An overview over application, network, and operating system level QoS parameters can be found in [8].

Traditional system-related service descriptions are not sufficient and need to be extended. Parameters applied at enhanced service interfaces include several QoS parameters (e.g. throughput, delay, jitter, reliability), general service parameters (e.g. multicast, synchronization, security) as well as a type of service (ToS), that describes qualitative how a QoS parameter should be guaranteed (deterministic, statistical, best effort).

QoS mapping performs an automated translation between different QoS specifications. Thus, one task of QoS mapping is the bidirectional translation between the application-visible QoS parameters at the application interface and application-transparent QoS parameters at system internal service interfaces. Mapping is additionally necessary between all internal service interfaces. This implies that traditional layered communication systems need one separate mapping function for each communication layer.

QoS negotiation distributes the service requirements to all involved entities. A local negotiation is performed between service user and service provider in a single host, whereas end-to-end negotiation comprises all communication systems that are passed by a data stream. End-to-end negotiation is typically performed at network level, where specialized reservation protocols are applied (e.g. similar to ST-II [9] or RSVP [16]). During QoS negotiation, *local resource managers* are activated in each communication system and QoS parameters are adapted based on the current resource capacity. Generally, QoS negotiation is performed in a two or three way handshake during connection establishment. After successful QoS negotiation, a *traffic contract* describing all negotiated QoS parameters is concluded. Typical forthcoming applications have highly dynamic characteristics, e.g., QoS parameters or the receiver group change in short time periods. In these scenarios *QoS re-negotiation* functions become very important.

To ensure the achievement of the requested QoS, *QoS maintenance* functions are needed. Mechanisms have to be identified that support the required QoS parameters. Additionally, the achieved QoS values have to be monitored during operation and possibly supporting mechanisms need to be adapted accordingly. *QoS monitoring* computes current values of QoS parameters and is able to detect QoS violations. For this task, a QoS monitor frequently compares current measured QoS values with the requirements initially agreed in the traffic contract. Thus, QoS monitoring plays an integral part in a QoS maintenance feedback loop [1].

3. Communication subsystems model

Communication subsystems enhance the network service by a set of protocol functions to achieve the service requested by the application. Service integrated communication systems must therefore be able to offer application-tailored services to a variety of applications. These services require appropriate protocols to efficiently support service requirements. Protocol functions that are not needed should be turned off, since they reduce the achievable performance [11]. Thus, an application-tailored protocol can be build by selecting the minimal possible set of protocol functions supporting the requested service. Moreover, specific protocol mechanisms implementing a protocol function (e.g. window-based or rate-based flow control) can be selected and parameterized dependent on the requested service.

Figure 1 shows a model of a communication subsystem that is capable to build application-tailored services. The model comprises a data path including all time critical data operations and a separate control path with control functions, such as QoS management. The data path is represented by the protocol entity consisting of a set of service specific protocol functions. The control path comprises a protocol control agent and a QoS monitor. The protocol control agent builds application-tailored protocols by selecting appropriate protocol functions based on the requested service. Besides the service requirements, several rules are needed that describe relations between QoS parameters and protocol functions or mechanisms [13]. Additionally, the protocol control agent interacts with a central QoS manager that co-ordinates all QoS related activities in the end system. Initially, the protocol control agent uses only service requirements to select protocol functions. *Current* QoS parameters continuously measured by the QoS monitor are used to improve the service by adapting protocol parameters or functions. Therefore, the protocol entity hands

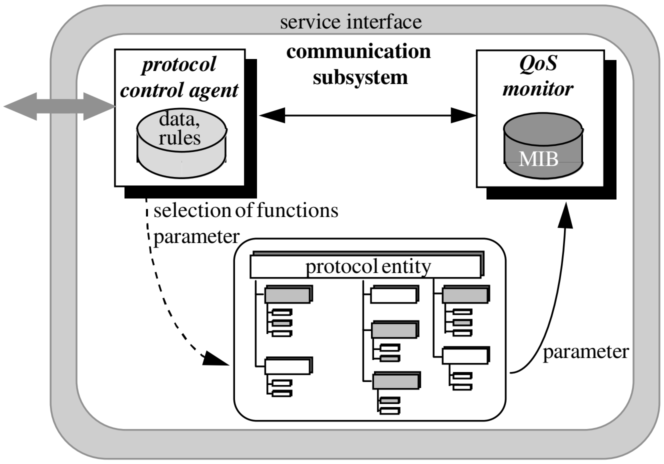

Figure 1: Model of the communication subsystem

values of protocol parameters over to the QoS monitor, which computes current QoS values and stores all relevant information in a QoS-MIB (Management Information Base). The QoS monitor informs the protocol control agent about significant changes in QoS values or about QoS violations. The latter one is responsible for further actions. Design and implementation of the QoS monitor are presented in the next section in detail.

4. QoS Monitoring in the Communication Subsystem

In general, a QoS monitor is responsible for collecting current values of QoS parameters and detecting QoS violations by comparing these values against specified limits. The definition used in this work states that QoS monitoring is concerned with the collection of traffic contract parameters and the reactions to parameter changes. Thus, two different parts of a QoS monitor can be identified. First, monitoring detects whether the application obeys the traffic contract. Second, resulting QoS parameters have to be monitored continuously and, additionally, violations of negotiated QoS parameters have to be detected.

4.1. Requirements

The QoS monitor has to fulfill certain requirements that can be derived directly from its definition and environment. Thus, the monitor

- should ideally not affect the performance of the monitored communication subsystem. Practically, one should try to minimize the resources needed by the monitor and never stop the data flow of the monitored protocol entity.
- should have a high accuracy and resolution. Reports of QoS violations should not be caused by inherent inaccuracies of the QoS monitor.
- must have a high performance, because it has to report QoS violations in time to avoid hysteresis effects.
- should be flexible to monitor concurrently several very different data streams which may vary their QoS demands over time.

4.2. Design decisions

There are certain alternatives for the design and implementation of a QoS monitor. Our decisions obey the requirements given above. They can be grouped around the architectural, performance, and functional aspects, discussed in the following.

Architectural aspects

Autonomous entity Placing the monitor in an autonomous entity results in a minimal influence on the protocol performance. The alternative, namely embedding the monitor into the protocol, would steal processing time from the protocol, because there is only one activity thread in which all computation must be done. Using two activity threads, i.e. processes, decouples the monitor from the protocol and gives the opportunity to let the protocol work with a higher priority than the monitor. Using asynchronous communication between both entities allows to them run independently and in parallel, i.e., the protocol has never to wait for the monitor. On the other hand, because of the tight coupling in time, the monitor should not go too far behind the protocol.

Location of the monitor Monitoring can be performed at the sender or the receiver side. Most QoS parameters are only measurable at the receiver (e.g. end-to-end delay, reliability). A single monitor at the sender side requires that the receiver sends all measured values (e.g. event-id, event type, timestamps) back to the sender [4]. This produces a high network load and delay in the evaluation of the QoS values. Multicast scenarios make this situation even worse. We decided to locate one part of the monitor at the sender side for monitoring the application traffic (i.e. sender throughput). This parts detects whether the application obeys the traffic contract. Additionally, a second part is located at the receiver side for measuring all QoS information available there. The part located at the receiver side is informed about the negotiated parameters of the traffic contract, and thus, is able to do perform evaluation and checking for violations by itself. Network traffic is reduced since only in the case of a QoS violation that seems to be caused by the sending entity or the network, an information packet is send back over the control channel.

In the end system itself, monitoring can be done either inside the communication subsystem at the network interface or at the service interface. Monitoring at the service interface is preferable since it allows a direct comparison with application requirements and QoS violations can be easily detected. However, the general monitor architecture can be used as well for protocol parameters. In this case, a mapping of application requirements to protocol parameters is required.

Performance aspects

Event driven monitor We decided to build an *event driven monitor*, instead of using a *sampling* method for collecting measurement data. Sampling, i.e., collecting data at certain constant time intervals (e.g. every 0.01ms) has two big disadvantages. First, the determination of the time interval is a crucial task, and second, this method leads to a non negligible overhead caused by the interrupt processing. Generally, a timer device is used for sending an interrupt every measuring period. For keeping up with the high data rates of high performance networks, the measuring period has to be very short. This requires a very high number of interrupts per second, leading to a high system load. Even if the protocol is idle, the system is under load because of unnecessary sampling.

On the other hand, the *event driven* approach avoids interrupts at all, instead protocol events trigger monitoring actions. During normal protocol processing, the protocol collects relevant information, reads the system clock (which can be done in one CPU cycle on modern workstations) and associates collected monitoring events with a time stamp. Afterwards it informs the monitor entity with a signal about the ready measurement data. By receiving such a signal the monitor starts to evaluate the signalled events in parallel to protocol processing. Generally, the protocol can separately issue a signal for each event, or alternatively, collect several events and signal them together. We decided to use an adaptive method that is based on the measured throughput and the clock resolution (cf. section 4.2.).

Functional aspects

Monitoring of simplex data streams We assume that the communication subsystem is based on simplex data streams (also called "flows") and additionally, a separate duplex channel for control data (signalling channel). Typically, data and control informations have highly different QoS requirements and thus, this separation leads to a more efficient usage of the network resources.

Monitored parameters In a first step, we decided to monitor the parameters throughput (in bytes), rate (in data units), data unit size, end-to-end delay, delay jitter, and errors (for lost and duplicated data units and checksum errors). Each of these parameters is described by a set of values stored in a so-called

QoS vector. More detailed, the QoS vector includes a minimum and maximum limit, a lower and upper threshold, an average value, an average interval (defining the number of consecutive data units over which the average is computed), a ToS class and a fractional bound. Typically, not all of these values are of interest for a QoS parameter and, therefore, it is possible that only subsets are specified.

Three types of service classes (ToS class) deterministic, statistical, or best effort, describe how the minimum and/or maximal limit should be guaranteed. A deterministic guarantee allows the specification of an interval, that defines hard bounds for all measured values (in some cases, only a single bound, i.e., the maximum or minimum value has to be guaranteed). Values outside the interval bounds are always regarded as QoS violations. Parameters of the statistical class are specified by an interval and a fractional bound $\frac{a}{b}$ as proposed in [5]. This bound specifies the number a out of b consecutive values that are allowed QoS violations. Checking a measured value against a fractional bound, requires to keep a history of b values of this parameter and to count all violations within these b values. Each time, a new value comes in, the longest stored value is "forgotten" and the violation-counter is updated accordingly. This so-called "sliding window" technique has some advantages in contrast to a "jumping window" technique which collects a set of b values and then "forgets" the whole history and starts from begin. In this case, violations that are located around the "window border" can be twice as high as originally negotiated without any violation indication. Furthermore, it is not possible to check continuously for the fractional bound. The last type of service class, the best effort class, will never produce a QoS violation, because there are no bounds wherein the parameter values of this class must lie.

Besides the hard limits, we use thresholds that define the begin of a critical area where QoS values have not yet reached the limit, but may reach it soon. Using thresholds can help the system to react earlier, e.g., to allocate additional resources. But normally, a reaction should only be initiated if the values tend monotonously towards the hard limit.

In the following, we describe by which techniques we measure QoS parameters at the moment.

- throughput or rate
  Assuming that a data unit p_i of size s_i is received at the time t_i, then the throughput (measured in bytes) of $k+1$ consecutive data units $p_i \dots p_{i+k}$ is calculated as $\theta_{i \dots i+k} := \sum_{l=i}^{i+k} s_l / (t_{i+k} - t_{i-1})$.

  In our method, each data unit is associated with a timestamp when it is received. A serious problem occurs, since exact times are rounded off by the clock hardware to an integral multiple of the clock resolution (quantization error). Some events may therefore get the same timestamps, resulting in a time difference Δt of zero, which is useless for computing throughput. If Δt is near clock resolution Δt_{min}, we nevertheless have a non negligible relative measurement error of $\epsilon = \frac{\Delta t_{min}}{\Delta t}$.

  We decided to collect data units until the time difference $\Delta t = t_{i+k} - t_{i-1}$ is big enough for guaranteeing a predefined maximal relative measurement error ϵ. It follows that we must collect data units until $t_{i+k} \geq t_{i-1} + \frac{\Delta t_{min}}{\epsilon}$. Afterwards, the monitor is starts to compute the throughput of this measurement interval as $\theta_{i \dots i+k}$.

  Bursty traffic with long silence periods can lead to a situation where k very fast data units arrive, but Δt is still less than $\frac{\Delta t_{min}}{\epsilon}$. Caused by a silence period, the protocol may wait a long time for the next data unit $k+1$ to arrive before it hands over the collected data to the monitor. It would then calculate a too low throughput for the interval of the first k data units. We avoid this inexact computation by checking always the time difference $t_{i+k} - t_{i+k-1}$. If this difference is greater than $\frac{\Delta t_{min}}{\epsilon}$, i.e. there was a silence period between the last two data units, we simply set $t_{i+k-1} := t_i + \frac{\Delta t_{min}}{\epsilon}$, calculate $\theta_{i \dots i+k-1}$ and a separate θ_{i+k}.

- end-to-end delay
  We measure end-to-end delay by exchanging timestamps and thus need synchronized clocks in all involved end systems. Synchronized clocks can be realized by using radio-controlled hardware clocks or a synchronization protocol like NTP [10]. Each time data is ready for transmission, the sender generates a UTC timestamp that is placed in the transmitted packets and stripped off by the receiver. This reveals the need of changing protocols which do not support timestamps at all or not at the needed resolution of nanoseconds. The receiver generates another timestamp as soon as the data is delivered. Based on this information, the monitor computes end-to-end delay as difference between the two time stamps.

- delay variation (delay jitter)
  For applications with a continuous data flow, such as video, it is desirable that the communication

system guarantees a maximum delay jitter J_{max}. We use the definition of [5] where the delay jitter J_i of one data unit p_i is defined as difference $|d_i - d|$ between a chosen *target end-to-end delay* d and the current end-to-end delay d_i this data unit. Violations of the maximum jitter bound are detected by checking the end-to-end delay of a received data unit against the minimum and maximum delay bounds. Thus, our monitor realizes the monitoring of jitter by computing the minimum and maximum end-to-end delay d_{min}, d_{max} of the last k data units and checks for the condition $d_{max} - d_{min} \leq J_{max}$.

- errors
  Dependant on the selected error detection and recovery functions, a protocol entity is able to report various error types to a monitor. Our monitor is able to count error values over a predefined interval (for traditional protocols the interval covers the whole connection duration). The four error types: lost data, corrupted or incomplete data, duplicated data, and indicated errors can be measured. Besides really lost data units, the parameter for lost data counts data units arriving too late for delay sensitive applications. Delayed data is for these applications no longer of interest and can be discarded. The counter for indicated errors represents the number of detected, but not recovered errors.

Reactions The collected parameters and possible QoS violations are information that is useful to other entities, e.g., the protocol agent or a network manager. How and which part of the information is propagated to other entities can be configured. A *monitoring policy* parameter defines possible reactions of the monitor and can be specified separately for each data flow. Furthermore, the monitoring policy is changeable during operation. Based on the monitoring policy, the monitor reports current QoS parameters periodically, answers to requests, issues QoS violations immediately or QoS warnings before QoS violations are detected. If a QoS violation is detected, the monitor reports it once. Afterwards it waits to allow for a reaction of the protocol agent. As soon as the protocol agent has indicated that it was able to cope with the problem, the monitor starts reporting violations again. Thus, informing the peer entity is not a task of the monitor, but of the protocol agent in co-operation with the QoS manager.

4.3. Monitoring Architecture

The overall monitoring architecture is shown in Figure 2. The QoS monitor comprises three entities, that are able to work independently. They are grouped around a QoS-MIB where all QoS related informations are stored. The core of the QoS monitor is a *monitor entity*, that is always active when the QoS monitor is activated. It is logically divided in two parts, a *communication module* for all communication with the protocol control agent and a *computation and evaluation module* that computes QoS parameters from protocol information, updates QoS parameters in the QoS-MIB and detects QoS violations. Both modules are described in detail in the sections 4.3. and 4.3.. The two other entities are a local *presentation entity* and

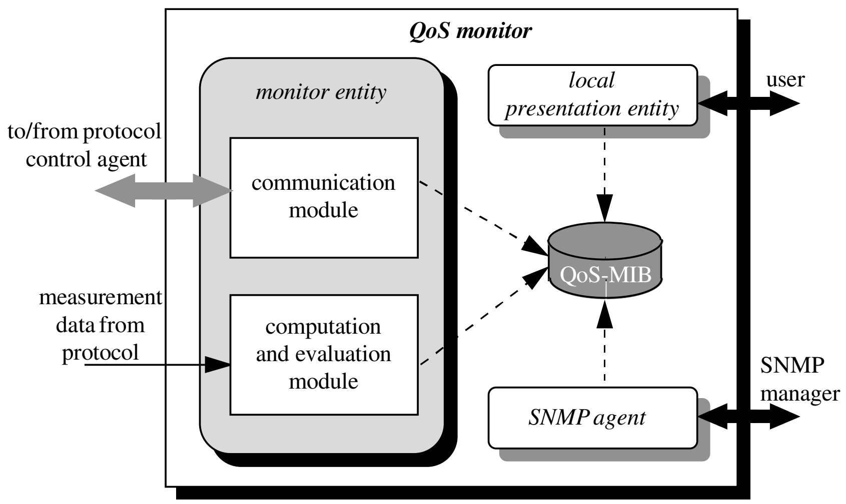

Figure 2: Monitoring architecture

an *SNMP agent*. The presentation entity presents local QoS parameters stored in the QoS-MIB graphically to a user. It is activated and totally controlled by a human user. Thus, the user is for example able to select an update frequency for the presented parameters [6], allowing a scalable tradeoff between system load and actuality.

The SNMP agent performs the integration of QoS monitor with traditional SNMP management [14]. An SNMP manager located at a remote system is therefore able to control QoS parameters of several systems or at different levels, e.g., at ATM or transport system level. Additionally, monitoring and comparison of service quality of receivers of a multicast connection is possible. Currently, the SNMP agent has only read access to the QoS-MIB.

Communication Module

The communication module is responsible for the whole communication with the protocol control agent (cf. Fig. 2). The latter is able to configure dynamically the monitoring entity and request current parameters. In turn, the communication module propagates QoS violations detected by the computation and evaluation module to the protocol control agent.

In the following, we describe some of the primitives the interface offers: To clearly identify one of several connections, the parameter `ConnectionID` is used by each primitive.

- `NewConnection(ConnectionID,QoSSpec,ReportPolicy)`
  `CloseConnection(ConnectionID)`
  The protocol control agent informs the monitor by the `NewConnection` primitive about a new connection. The `QosSpec` parameter includes all values of the QoS vector, the `ReportPolicy` parameter specifies the reactions of the monitor entity (cf. paragraph "Reactions" in section 4.2.). The `CloseConnection` primitive indicates that the specified connection is closed and the monitor can free all used resources from this connection.

- `ChangeQoS(ConnectionID,QoSSpec)`
  `ChangePolicy(ConnectionID,ReportPolicy)`
  With the `ChangeQoS` primitive the protocol control agent changes the QoS specification for the particular connection (e.g. used in case of a QoS renegotiation). The `ChangePolicy` primitive alters the report behavior of the monitor.

- `GetQoSParam(ConnectionID,QoSParamList)`
  `GiveQoSParam(ConnectionID,QoSParamVector)`
  The protocol control agent requests measured values of QoS parameters listed in `QoSParamList`. Current values are reported in a QoS vector `QoSParamVector` by the `GiveQoSParam` primitive from the monitor. Additionally, this primitive is issued periodically, if the appropriate reporting policy is selected.

- `QoSReport(ConnectionID,Type,Report)`
  QoS violations and warnings are indicated from the monitor by a `QoSReport` primitive. `Type` specifies whether a violation or a warning is reported. The `Report` parameter identifies clearly the concerned parameters, their values and the reason for this indication.

Computation and evaluation module

The computation and evaluation module is able to handle simultaneously several QoS parameters of multiple connections. The computation part derives QoS parameter values from collected protocol parameter values and updates QoS values stored in the QoS-MIB, accordingly. The evaluation part checks the calculated QoS values against their limits given by the traffic contract. It informs the protocol control agent about QoS violations. The protocol hands measurement data over by the primitive `GiveMeasurementData(ConnectionID,Table)`. The (collected) protocol values for one or more data units are stored in a table, given by parameter `Table`.

4.4. Implementation

Our implementation monitors QoS parameters at the transport service interface. Thus, all parameters are related to service data units (SDUs). We implemented the QoS monitor under the operating system OSF/1 on DEC ALPHA stations. It works together with the transport protocols PATROCLOS [3] and XTP-Lite of the BERKOM project [6].

The monitor and the protocol run in the same address space (so called light weight processes or *threads*) using semaphores, asynchronous signals and multiple buffering for minimizing the use of resources and operating system overhead [2]. The monitor thread works with a lower priority than the protocol thread, leaving more CPU time for protocol processing.

The protocol was enhanced to transmit 64-bit timestamps with each SDU. We use a 32-bit field for the seconds and a 32-bit field for the nanoseconds. The ALPHA workstation is able to read the system clock in only one CPU cycle and has a maximum resolution of 1/1024 second. For clock synchronization we use a public implementation of the Network Time Protocol (NTP) [10].

The protocol manages currently two tables for each simplex connection including SDU number, SDU size, timestamps, and error type in a table row. The monitor reads from one table while the other one is locked by the protocol for writing (double buffering). The communication between monitor and protocol is realized with simple FIFO-queues in shared memory. If a table is ready for computation, the protocol places a message into the queue and the monitor copies the values from the table into another private table. It continues to compute QoS parameters per SDU and finally per connection. In dependence on the ToS class QoS parameter values of each SDU are checked for violations (evaluation part).

In the past, the monitor was mainly tested with the XTP-Lite protocol of the BERKOM project. Throughput values of several Mbit/s were measured by the monitor during the transmission of a video data stream. These values were limited by the prototypical implementation of the protocol. We plan to test the monitor in scenarios with higher data rates. Currently, SandiaXTP [12], an object-oriented implementation of XTP 4.0 [15], is enhanced with an ATM interface and it is planned to perform tests with this XTP implementation in a local ATM testbed as well as wide area tests over the ATM testbed of Deutsche Telekom.

5. Summary and Future work

Generally, QoS monitoring is one important task in the QoS management field. Especially, QoS maintenance functions depend on the information collected in a QoS monitor. In this paper we have presented the design and a prototypical implementation of a flexible monitor for high performance communication subsystems. The monitor can be configured to measure several QoS parameters and, additionally, detect QoS violations based on the selected ToS class and specified limits. The whole monitoring architecture is embedded in an SNMP based management framework and provides a graphical interface to a human user. Currently, the monitoring interfaces are enhanced to allow a flexible operation with several protocols. Furthermore, we are working on the integration of the monitor with other components, such as the presented protocol control agent.

References

[1] Aurrecoechea, C.; Campbell, A.; Hauw, L.: *'A Survey of Quality of Service Architectures'*; Technical Report MPG-95-18, Lancaster University; 1995

[2] Bless, R.: *'Entwurf und Implementierung eines Monitors für Dienstqualitäten'* ; Studienarbeit; Universität Karlsruhe; Institut für Telematik; Fakultät für Informatik; August 1995

[3] Braun, T.; Schmidt C.: *'Parallel transport subsystem implementation for high-performance communication'*; Concurrency: Practice and Experience; Vol 6(4), pp.375-391; June 1994

[4] Campbell, A.; Coulson, G.; Hutchison, D.: *'A Quality of Service Architecture'*; ACM Computer Communications Review; Volume No. 24, Issue 2; pp 6-27; April 1994

[5] Ferrari, D.: *'Client Requirements for Real-Time Communication Services'*; IEEE Communications Magazine; Vol. 28, Nr. 11; pp 65-72; November 1990

[6] Fischer, K.-U.: *'Überwachung von Dienstgüteparametern in XTP-Lite'*; Studienarbeit; Universität Karlsruhe; Institut für Telematik; Fakultät für Informatik; September 1995

[7] Hutchison, D.; Coulson, G.; Campbell, A.; Blair, G.: *'Quality of Service Management in Distributed Systems'*; Network and Distributed Systems Management; Editor: Morris Sloman. Addison Wesley; pp 273-302; 1994

[8] Nahrstedt, K.: *'An Architecture for End-to-End Quality of Service Provision and its experimental validation'*; Dissertation in Computer and Information Science; University of Pennsylvania; Philadelphia; 1995

[9] Topolcic, C. (Editor): *'Experimental Internet Stream Protocol, Version 2 (ST-II)'*; Request For Comments 1190; Oktober 1990; IETF

[10] Mills, D.L.: *'Network Time Protocol (Version3) – Specification, Implementation and Analysis'* ; Network Working Group; University of Delaware; Request For Comments 1305; IETF

[11] Richards, A.; Seneviratne, A.; Fry, M.; Witana, V.: *'Tailoring the Transport Protocol for Giga Bit Networks'*; Australian Telecommuncation Networks and Applications Conference; December 1994

[12] Sandia National Laboratories: *'SandiaXTP User's Guide – SandiaXTP - An Object-Oriented Implementation of XTP 4.0'* Distributed Systems Research; Sandia National Laboratories; Livermore; California; USA; May 1995

[13] Schmidt, C.; Zitterbart, M.: *'Towards Integrated QoS Management'*; Proceedings of the 5th IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems; Cheju Island; Korea; August 1995

[14] Seitz, J.: *'Netzwerkmanagement'*; Thomson's Aktuelle Tutorien; Nr. 2; International Thomson Publishing; Bonn 1994 Request For Comments 1305; IETF

[15] XTP Forum: *'Xpress Transport Protocol Specification – XTP Revision 4.0'*; XTP Forum; Santa Barbara; California; USA; March 1995

[16] Zhang, L.; Deering, S.; Estrin, D.; Shenker, S.; Zappala, D.: *'RSVP: A New Resource Reservation Protocol'*; IEEE Network; September 1993