Chatroom over WLAN: Systematical Development of a QoS-Integrated Distributed System

Jens Brandt, Reinhard Gotzhein, Rüdiger Grammes and Bernd Schürmann

Department of Computer Science
University of Kaiserslautern
67653 Kaiserslautern, Germany
Email: {j_brandt,gotzhein,r_gramme,schuerma}@informatik.uni-kl.de

***Abstract*: WLAN technology is an interesting alternative to usual local area networks. Due to its sensitivity to noise, realizing applications with QoS requirements on the basis of this technology is a challenging task. In this paper, we present the development of a chatroom over WLAN, focusing on QoS aspects. In particular, we address the selection of suitable and sufficient QoS-mechanisms on the application, middleware, and basic service layers, and their harmonization across layers.**

1. Introduction

WLAN is a popular standard for wireless ad-hoc communications. WLAN communication has several advantages. It enables mobility and portability. Installation costs can be reduced since no cable infrastructure is needed. However, the noise on the medium can vary dynamically, and can become very high. For example, Bluetooth cards and microwaves can interfere with WLAN. Therefore, error correction and dealing with heavily varying channel qualities is much more important to WLAN communication than to other cable-based communication technologies.

Using high-speed WLAN cards in modern buildings enables a variety of applications, for instance, *building control* needing very short reaction times, *file transfer* via Internet with high average throughput, and *audio* and *video communication* based on isochronous data streams. Each of these applications requires a specific quality of service in terms of reliability, throughput (peak, average, minimum), latency, and jitter, as well as certain service guarantees (deterministic, statistical, best effort).

Due to the varying channel quality, deterministic guarantees for WLAN communication are beyond reach. However, statistical guarantees are feasible in principle, which is sufficient for many applications. For each application, the following problem has to be solved: propagate the QoS provided by the WLAN technology up to the application level, while transforming it into the QoS required by the application, and vice versa. For instance, a raw throughput in bps on a TDM/FDM medium may have to be mapped to a video data stream with a fixed frame rate and an upper bound for latency and jitter on the application level.

An approach to solve the stated problem is to develop a small number of *general-purpose protocol stacks*, supporting a wide range of applications on top of different basic technologies, as done, for instance, in the Internet. However, due to the wide spectrum of applications, the communication services offered by these protocol stacks will not always be adequate. In these cases, *special-purpose protocol stacks* may be developed in addition. This approach offers the opportunity to fully exploit the properties of the basic technology, and to define a protocol stack that is customized w.r.t. a particular application, or a class of applications with similar communication requirements. These applications have to be modified to benefit from the special purpose stacks.

In this paper, we report on the development of a *chatroom system* based on *WLAN technology*, focusing on *QoS aspects*. The bandwidth needed by such a system is low compared to the average bandwidth provided over the WLAN basic service, but with several applications competing for the medium, the available bandwidth can still be a bottleneck. On the other hand, the chatroom system is easy to implement and serves as a prototype for systems with higher requirements on bandwidth and delay. Further work will focus on adding audio support to the chatroom system.

The chatroom system consists of a graphical user interface, application components, a customized communication middleware, and a QoS-oriented WLAN driver. Emphasis has been on the systematical development, and on QoS-integration. By *systematical development*, we refer to the methodological aspects, for instance, the structuring of the system according to the QoS architecture [1], and the use of description techniques in all phases. *Qos-integration* addresses the selection of suitable and sufficient QoS-mechanisms on each level of abstraction (application, middleware, basic service), and their harmonization across system layers.

The chatroom system has the following features:

- Users are admitted to the chatroom only if a minimum rate of chat messages can be statistically guaranteed (*QoS provision*).
- Access to the chatroom is granted on the basis of the current weighted average transmission rate of the WLAN medium (*feedback loop*) and the minimum rate of chat messages.
- For each chat user, the maximum rate of chat messages is dynamically adapted to varying channel quality, channel usage, and chat frequency (*QoS management*).
- Chat messages can only be sent within the limits of the current traffic contract (*QoS control*).

The paper is structured as follows: Section 2 surveys the main concepts of QoS architectures. Section 3 describes

the overall architecture of the developed chatroom system, and the functionalities of the system layers. Section 4 shows how the QoS concepts outlined in Section 2 have been integrated into the system. In Section 5, we draw conclusions.

2. Quality of Service Concepts

In this section, we identify the main concepts found in current QoS architectures [1, 5] and describe them briefly. We start with the constituents of a QoS specification, and then address the various functionalities of distributed systems supporting QoS in one way or another.

2.1. QoS specification

The QoS specification is part of a traffic contract between a set of service users and a service provider. Depending on the kind of service, the concrete parameters of the QoS specification may vary. For instance, quality of service can be defined from the distributed application point of view, which involves resources and QoS attributes as perceived by the user. Or, it can be defined from the hardware point of view, addressing concrete resources such as communication media, memory, and CPUs. The QoS specification – independent of the level of abstraction – usually addresses the following aspects:

- *performance*: quantitative aspects of the traffic contract (e.g., peak and average throughput, burst characteristics, delay, jitter, loss rate, corruption rate)
- *synchronization*: time relationship between different streams (e.g., lip synchronization for audio and video streams)
- *QoS guarantee*: binding character of the traffic contract (e.g., deterministic, statistical, best effort)
- *QoS management policy*: scalability aspects (e.g., reduction of frame rate or resolution in case of deteriorating throughput)

Depending on the kind of service, only some of these aspects may be relevant, while others are neglected. Also, depending on the service provider, only some of these aspects may be actually supported. For instance, in the Internet, only best effort guarantees can be given.

2.2. QoS provision

Before a service can be provided with a certain quality, a traffic contract is negotiated. Based on the QoS specification, the service provider checks the resource situation, and allocates resources. This requires a QoS mapping, admission control, and the coordination of local reservations:

- *QoS mapping*: The objective here is to translate the QoS specification supplied by the service user to the resource view of the service provider, i.e., to associate resources of different levels of abstraction and to determine corresponding QoS parameter values. For instance, throughput in terms of the application may be measured in video frames per second and resolution, with further requirements on delay and jitter, and is then mapped to periodicities, deadlines, and processing times for processor and communication medium, and possibly the size of a playout buffer. As the service provider may itself be refined into protocol entities using a lower service, QoS mapping may occur several times.
- *reservation protocol*: To provide a communication service with a specified quality of service, virtual connections are established and maintained. In particular, routes are determined such that the QoS requirements of the service users are met. Moreover, the local reservations are coordinated. Further aspects covered by the reservation protocol may be the handling of connections with dynamic QoS requirements, the support of multicast connections, or the dynamic establishment of alternative connections due to changes in connectivity.
- *admission control*: To accept a new service request, the resource situation has to be checked, based on the QoS specification obtained after the QoS mapping. This is addressed by an admission test, which depends on the kind of resources, the scheduling strategies and the QoS guarantees. Part of the admission test may be, for each resource, a schedulability test. Furthermore, admission control covers the reservation and release of local resources and the dynamic adaptation of existing reservations. Also, distributed resources such as communication media can be taken into account.

It should be pointed out that admission control and the coordination of reservations may occur on several levels of abstraction. For instance, some decisions may be taken on the application level, while further decisions are postponed until the communication middleware or even the basic technology.

2.3. QoS control

If a service request is granted, a traffic contract is established between the service user and the service provider, with obligations for both parties. QoS control refers to short term functionalities applied by the service provider to fulfill his traffic contracts:

- *traffic policing*: To be able to fulfill his part of a contract, the service provider may wish to compare the actual traffic with the agreed traffic (traffic control). If the actual traffic exceeds the agreed traffic, the service provider may apply measures to adapt it (traffic shaping), for instance, by discarding or postponing data.
- *scheduling discipline*: The purpose of the scheduling discipline is to control the competition of service request units (e.g., messages) for a particular resource by applying certain preference rules. To provide deterministic guarantees, for instance, rate-monotonic scheduling or earliest deadline first are well-known disciplines.

To ensure that the number of competing request units can always be served, schedulability tests are carried out

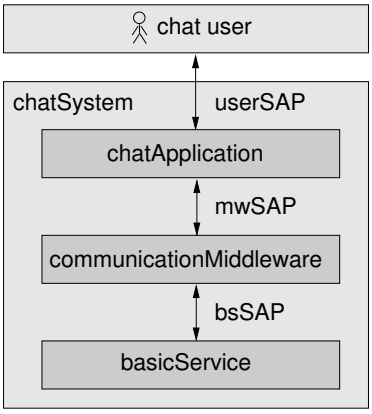

Figure 1: Structure of the chatroom system

as part of the admission test (see Section 2.2.). Furthermore, traffic shaping techniques may be applied.

2.4. QoS management

QoS management refers to long term functionalities applied by the service provider to fulfill his traffic contracts. These functionalities are needed if the network load can exceed the network capacity, for instance, in case of statistical guarantees or best effort, and support the adaptation of QoS as well as the improvement of network performance. The following functionalities can be distinguished:

- *QoS monitoring*: Here, the current performance parameter values of the network (e.g., throughput, delay, jitter, loss rate) are determined.
- *QoS maintenance*: This covers all kinds of tuning measures to fulfill current traffic contracts, and is based on the network status.
- *QoS scaling*: To cope with deteriorating network performance and increasing service requests, several approaches can be considered. For instance, QoS filtering techniques can be applied to reduce network load, for instance, in order to adapt data streams in group communication scenarios. Also, the sender may adapt his traffic to changing network characteristics.

While QoS provision and QoS control are usually associated with deterministic or statistical guarantees, QoS management is also useful in case of best effort networks.

3. System design

Before going into QoS details, we describe the architecture and general design of the chatroom system. The system is divided into three layers (see Figure 1):

- *chat application*: The application provides the graphical user interface for the chatroom system, handles user input, and displays chat messages and other information.
- *communication middleware*: The middleware is the core of the chatroom system. It manages the chatrooms, distributes the chat messages, and provides the central resource management functions. The middleware is customized to the application, and the WLAN basic service.
- *basic service*: The basic service adapts a basic technology (WLAN). It controls the hardware and medium access and gives feedback on the current resource situation (link quality, load, etc.). It offers service primitives to send and receive frames reliably to one or more receivers (multicast support).

3.1. Application

The application consists of the user interface userSAP and the chatApplication. userSAP is the interface over which the chat user interacts with the chat system. It takes input from the user and displays information and messages, e.g. information about the currently active chatroom and the users subscribed to this chatroom. It consists of the following elements:

- A dialog for entering a username.
- A text window for displaying chat messages and events.
- An input-line for entering commands and chat messages.
- A user-list of the currently active chatroom.

The chatApplication parses user input, extracts commands from the user and performs the necessary calls to the middleware interface. It also handles messages and events coming from the middleware over the middleware interface. The user commands are as follows:

- *open(name,description)*, *close(name)*: opens (closes) a chatroom with a *name* and a *description*.
- *list()*: lists all open chatrooms with their description.
- *join(name)*, *leave()*: the user joins (leaves) the chatroom *name* and the user-list is filled with the participants of this chatroom (or is cleared).
- *send(msg)*: *msg* is sent to the active chatroom.

3.2. Middleware

The middleware provides all functions to manage chatrooms. After a successful *login*, the chat application can access the following service primitives at the mwSAP interface:

- *openCR(cr,desc)*, *closeCR(cr)*: A chatroom with name *cr* and description *desc* is opened (is closed).
- *listCR()*: A list containing all open chatrooms and their descriptions is returned.
- *joinCR(cr,qos)*, *leaveCR(cr)*: Join (Leave) chatroom *cr* with QoS *qos* (see Section 4.1.).
- *sendToCR(msg)*: Send the chat message *msg* to all application entities of this chatroom.

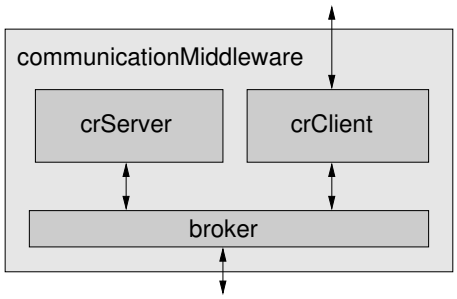

Figure 2: Structure of the middleware

Moreover, the application is notified when certain events occur:

- *crClosed()*: The current chatroom has been closed by its administrator.
- *userJoined(usr), userLeft(usr)*: The user *usr* has joined (has left) the chatroom.
- *msgReceived(msg)*: A new chat message *msg* has been received.

The communication middleware consists of three parts (see fig 2): the connection to the application appEntitySet, the connection to the basic service btCodex, and in between the part which contains the actual functionality, the coreMiddleware.

appEntitySet accepts new connections from applications, and creates proxies, over which the further communication with the core middleware is processed. It checks the users' identity and controls the access to chatSystem. btCodex deals with specialties that the use of a basic service involves. These are e.g. segmentation, reassembly, and lower layer setup. The block interacts with the basic service by means of service primitives.
The core middleware consists of three sub-blocks crServer, crClient, and broker.

The administration of the chatrooms is distributed over a chatroom server and its clients. The server controls the chatroom names. It maintains a list of created chatrooms and ensures that names are unique. Moreover it knows their description, their administrator, the names of subscribed users, and how to contact the chatrooms. So, clients requesting this information can contact the server, avoiding to ask all other clients. The clients, on the other hand, accept requests from the chat application entities (*openCR*, *closeCR*, *joinCR*, *leaveCR*) and send them to the server. The server notifies affected clients when the information changes, e.g. a user leaves a chatroom. The transmission of chat messages (*sendToCR*) is done by the clients themselves, without participation of the server. The clients maintain a list of local end points of each chatroom and take over the distribution of incoming messages.

While the client part is active in all middleware entities, the server is activated in exactly one of the network nodes. After starting of one or more middleware instances, the entity which will provide these functionalities is elected.

The broker provides a convenient abstraction layer for the communication between client and server. To contact the server, clients do not need to know the location of the server, the broker undertakes the task of distributing the messages in both directions.

Each chatroom is mapped to a multicast address of the basic service, which is requested during creation of a chatroom. So, joining (and leaving) a chatroom is done by joining (and leaving) the corresponding multicast group. Sending a message to a chatroom is done simply by sending the message to the multicast address.

3.3. Basic service

The basic service is responsible for the transmission of messages between the network nodes. Generally, it is possible that several middleware instances connect to a single basic service instance.

- *get_unicast(address)*, *put_unicast(address)*: To send and receive messages, each communication middleware must initially request an address (*get_unicast*). Releasing the address is achieved by *put_unicast*.
- *get_multicast(address)*, *put_multicast(address)*: The basic service allows multicast communication, too. Initially, a multicast group is created by allocating an unused multicast address (*get_multicast*), which is released after its use (*put_multicast*).
- *(un)register_multicast(address)*: All stations that want to receive the messages of the group have to subscribe.
- *send_frame(r_address,s_address,data)*: The basic service provides this service primitive to send a message. All receivers are notified by calling their *receive_frame* callback function.

The basic service is reliable, i.e. transmitted messages eventually arrive at the receiver without corruptions. The error and loss control is directly obtained from the collision detection of the medium access on this layer, which is similar to the protocol described in the IEEE 802.11 WLAN standard [4].

The basic service offers QoS monitoring functions, which are used by the communication middleware. These functions describe the link quality and current load of the medium. *poll_bandwidth* returns the average bandwidth. Every time a frame is sent, its sending delay is reported to the basic service users by *qos_feedback*. The basic service uses adaptive error correction to optimize the transmission quality using rate-compatible codes for channel coding [3].

4. QoS Components and QoS Mapping

The system that has been described so far lacks support for QoS mechanisms. This section describes how components have been added to fulfill the QoS requirements. To test different strategies and to benefit from reuse techniques, these components have been designed as *micro protocols* (i.e. self-contained protocols with a small functionality [2]).

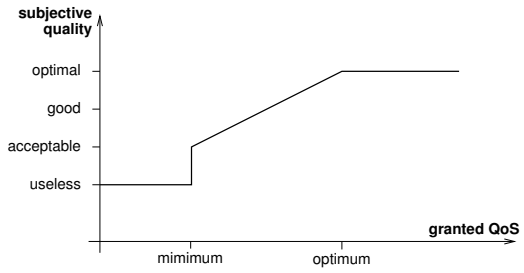

Figure 3: Subjective quality

The core QoS functionality is located in the communication middleware. The basic service supports it by offering monitoring functions. The application, in turn, uses the QoS service primitives of the middleware. Since the different layers in the system are tightly integrated, an unnecessary duplication of QoS components is avoided.

Designing the QoS components, we have to take into account the properties of the WLAN technology. Unlike other cable-based technologies, the total bandwidth of the medium changes with time due to noise. Therefore, deterministic guarantees cannot be given. Rather statistical guarantees based on an average transmission rate are feasible. Reservations have to be granted based on an average, not on a nominal transmission rate.

Furthermore, for each user, a minimum and an optimum transmission rate are defined. The reservation is made on the basis of the minimum rate, because as many users as possible should be supported. Dynamically, the QoS is scaled: If the medium has free capacity, each user can get up to the nominal transmission rate. If the capacity deteriorates (due to noise), then the transmission rate is reduced even below the reserved rate. The user gets feedback on any QoS change.

In the chatroom system, a certain QoS is requested by specifying an upper limit u and a lower limit l of messages per minute. The meaning of these bounds is as follows: a user needs at least the lower bound l - giving him less than l is useless. On the other hand, the user does not need (and cannot use) more than QoS u. In the chatroom system, these bounds result from a minimum rate of messages that is needed for a conversion without unbearable chat message delays (lower bound), and the maximum typing speed of the user (upper bound). In other applications like audio communication, these two bounds can be defined, too: the least QoS l to transmit with a sufficient audio quality to understand the conversational partner and the QoS u where no quality improvement can be noticed (Figure 3).

Using a wireless medium for the communication between the nodes, we have to address some problems as said in the introduction. Different from other basic technologies, the WLAN service only gives statistical guarantees, because unpredictable noise can reduce the QoS or even prevent the whole communication. So, strategies to react to QoS violations caused by disturbances on the medium have been defined on all layers of the system.

The system reacts by scaling the QoS of the users respecting their upper and lower bounds. Under nominal conditions, everybody is granted at least the resources minimally needed. If this cannot be provided, some users are notified and temporarily withdrawn all resources.

Table 1: QoS specs on different levels

layer	requested QoS	granted QoS
user	(un)experienced	red,yellow,green
application	#msg/min (min. and opt.)	#msg/min
middleware	mwQoSClass (see Table 2)	#frames/min
basic service	#bytes/sec	#bytes/sec

Table 2: QoS classes on middleware level

mwQoSclass	param.	description
constant	min	constant *min* frames/min
variable	min	minimal *min* frames/min
	max	maximal *max* frames/min

Continuously, the chat system gives feedback about the granted QoS. At the user level, the QoS is described by one of following three classes: *Green* means that the upper bound can be guaranteed, *yellow* will be signaled if the granted QoS is in between the specified lower and upper bound. *Red* means that not even the lower bound of QoS can be granted and the user is set to read-only mode.

4.1. QoS specification

All QoS mechanisms rely on an adequate QoS specification. In the chatroom system, two things must be described: The first one is the QoS requested by a chatroom user. On user level, this is done by distinguishing between experienced and unexperienced users. The other one is the QoS that is granted by the basic service. Depending on available resources, the user is shown a red, yellow or green QoS indicator (as described above). Other software layers use other levels of abstraction and therefore use other QoS specifications (which requires qos mapping, see Section 4.2..1). Table 1 sums up the QoS specifications which are used at the various layers of the chatroom system.

The QoS class of the middleware *mwQoSClass* can be chosen from Table 2. QoS class 'constant' specifies a single, minimum number of frames per minute, while QoS class 'variable' allows the specification of the minimum and the maximum number of frames per minute.

4.2. QoS provision

4.2..1 QoS Mapping

As the different parts of the system use different QoS specifications (see Table 1), functions are needed which map between the different abstractions.

user $\leftrightarrow$ application The mapping between user level QoS (requested as well as granted) and application level QoS is performed according to Tables 3 and 4. Table 3 describes what number of chat messages per minute (minimum and optimum) the application associates with

Table 3: QoS Mapping of user requests

user level	applic. level minimum	applic. level optimum
experienced	2 msg/min	4 msg/min
unexperienced	1 msg/min	2 msg/min

Table 4: QoS Mapping of system feedback

middleware	application	user
granted $<$ min	red	read only
min $\leq$ granted $<$ opt	yellow	minimum
granted = opt	green	optimum

an experienced or unexperienced user; Table 4 specifies how the system feedback is mapped to user feedback depending on the granted number of chat messages per minute.

application $\leftrightarrow$ middleware The middleware translates the application requests to its QoS classes regarding the size of messages and frames (data unit which can be transmitted by the basic service). For this, the mwQoSClass 'variable' (see Table 2) is used, since it allows a minimum and a maximum number of frames per minute (corresponding to the minimum and optimum number of messages on application level) to be specified. In the other direction, the granted QoS is mapped from frames per minute to chat messages per minute (see Table 1) according to the following formula:

$$\frac{\#msg}{min} = \frac{\#frames}{min} \cdot \frac{1}{\frac{\#frames}{msg}}$$

$\frac{\#frames}{msg}$ is calculated from the recent traffic.

middleware $\leftrightarrow$ basic service The mapping between middleware and basic service QoS is calculated in the same way:

$$\frac{\#frames}{min} = 60 \cdot \frac{\#bytes}{sec} \cdot \frac{1}{\frac{\#bytes}{frame}}$$

4.2..2 Reservation protocol

The crAdminServer and the crAdminClient (see Figures 4 and 5) are the core parts of the middleware. They realize the functional requirements of the chatroom system described in Section 3.

Moreover, they perform the coordination of the QoS reservations and handle the dynamic QoS requirements. E.g. if a new user joins a chatroom, the crAdminClient will issue a QoS request to the central crAdminServer which will forward it to the admissionControl.

4.2..3 Admission control

Looking at the requirements, every user of the chatroom system is statistically guaranteed a number of messages depending on his experience. So, before join requests from new users are accepted, its has to be checked whether this guarantee can be fulfilled under the current resource situation. Therefore, a global admission control is added to the server (see Figure 4).

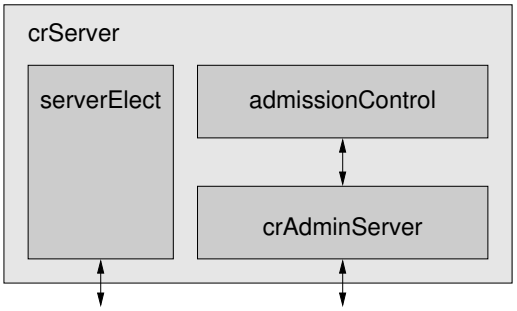

Figure 4: Structure of the chatroom server

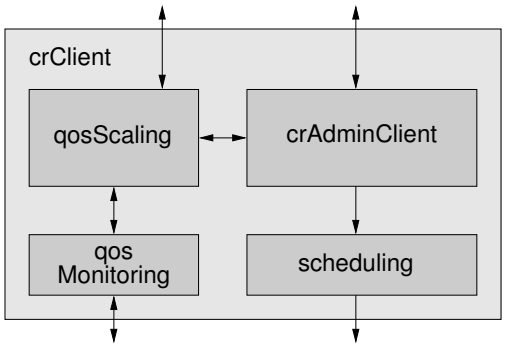

Figure 5: Structure of the chatroom client

Admission is granted depending on the remaining bandwidth of the basic service, which is calculated from the average bandwidth (obtained by *poll_bandwidth*) and the already reserved bandwidth.

4.3. QoS control

4.3..1 Traffic policing

Application and middleware negotiate on the number of messages each user can send to the chatroom. To fulfill its part, the application compares the actual number of messages with the agreed number. This is controlled by a traffic policing process inside the application, which non-command messages must pass.

Currently, the leaky bucket algorithm is used to control the traffic [6]. Each user may send a certain number of chat messages corresponding to his QoS value. This is realized by increasing a counter for every message sent, and decreasing it periodically after a fixed time span has elapsed. If the number of non-command messages exceeds the number of messages guaranteed by the traffic contract, the additional messages are queued and sent at a rate that conforms the contract with the middleware. This means that command messages are able to overtake non-command messages.

4.3..2 Scheduling discipline

Service request units that have passed the QoS control must be scheduled according to their priority. Since there is only one application with homogeneous QoS-requirements, i.e. all messages have the same priority,

the service request units can be scheduled in order of arrival (first come first served). In case of applications with heterogeneous QoS-requirements, this scheduling strategy is no longer sufficient.

Two user classes (experienced, unexperienced) exist on user level. Since experienced users may send chat messages at a higher rate than unexperienced users, a rate monotonic scheduling strategy could be used to give chat messages arriving in shorter intervals a higher priority.

4.4. QoS management

4.4..1 QoS monitoring

Although every user is guaranteed a certain transmission rate by the admission control, there are situations in which the throughput of the basic technology temporarily is less than required. To deal with this effect, the communication middleware has to detect it. This is done by monitoring the send delays of the basic service, that is the time difference between the send request and the sending of the message, using its *qos_feedback* function. If the traffic exceeds the capacity of the medium, frames will be queued in the basic service and the send delay is high. If there is little traffic, frames can be sent more or less immediately and the send delay is low.

In order to measure the send delay even when there is currently no traffic, qos monitoring frames are sent periodically.

As reservations have to be granted based on an average (avg), not on a nominal (nom) transmission rate, the basic service regularly determines the average rate by dynamic measurement (*poll_bandwidth*).

4.4..2 QoS scaling

The QoS scaling is completely performed in the communication middleware. Depending on the feedback described above, a local closed-loop feedback strategy calculates the QoS of the users, which must adapt their traffic. To assess the situation, the boundaries of the interval in which the send delays normally lie (provided that all users send their requested number of messages and noise does not disturb the communication) are calculated.

When a user joins a chatroom, the chat application requests a minimum and optimum number of chat messages per minute. Initially, he is given the minimum number. Every 30 seconds, the send delays collected by the qosMonitoring are evaluated: if they are outside the calculated boundaries, the QoS of one user is adapted. If the send delays are too high, the allowed number of messages for the selected user is decreased. If they are near the optimum, it is increased, respectively. This adaptation tends to be fair, i.e. users with low QoS are first granted more resources.

5. Conclusions

We have presented the systematical development of a QoS-integrated chatroom over WLAN. Starting from a set of requirements, for instance, concerning the throughput of chat messages, we have designed a layered system with QoS components on each layer by introducing self-contained QoS micro protocols, each of them dealing with a certain QoS aspect (e.g., QoS mapping, admission control, QoS scaling). Due to the nature of WLAN technology, a closed loop strategy that dynamically adapts to changing QoS has been followed. The resulting system is highly modular, with QoS components that can be replaced to implement different QoS mechanisms.

The implementation of the chatroom meets all specified requirements. In particular, the QoS requirements are satisfied during nominal operation of the basic service: throughput and end-to-end delay remain within the specified boundaries. If the medium cannot satisfy the traffic contract due to environmental conditions, the number of supported chat users is temporarily reduced. Under all conditions, the chatroom system grants the minimum QoS to as many users as possible.

The next goal is to add audio support to the chatroom system and to enable the routing of messages over different networks (e.g. the internet), in order to make the system accessible for a higher number of users. For future versions of the chatroom system, the integration of other basic technologies (like powerline) is conceivable. Another aspect is the support of different applications. For this, the middleware has to be extended to control the competition of service request units from different applications. It will be interesting to see how these new requirements fit into the system structure developed so far. We believe that further QoS functionality can be integrated in a straightforward manner into the given system. Further research will address the definition of a QoS component framework and a QoS microprotocol pool.

REFERENCES

[1] A. Campbell, C. Aurrecoechea, and L. Hauw. A review of QoS architectures. In *Proceedings of the 4th IFIP International Workshop on Quality of Service*, Paris, France, 1996.

[2] Reinhard Gotzhein, Ferhat Khendek, and Philipp Schaible. Micro protocol design: The SNMP case study. In *Proceedings of the 3rd SAM Workshop*, Aberystwyth, June 2002. SDL Forum Society.

[3] J. Hagenauer. Rate-compatible punctured convolutional codes (RCPC codes) and their appplications. *IEEE Transaction on Communications*, 36(4):389–400, April 1988.

[4] Wireless LAN medium access control (MAC) and physical layer (PHY) specification, 1997.

[5] F. Rößler and B. Geppert. Applying quality of service architecture to the field-bus domain. In *Proceddings of the 2nd IEEE International Workshop on Factory Communications Systems (WFCS'97)*, Barcelone, Spain, 1997.

[6] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, Englewood Cliffs, 2003.