A Framework for QoS Updates in a Networking Environment

Burkhard Stiller *
University of Cambridge, Computer Laboratory
New Museums Site, Pembroke Street
Cambridge, CB2 3QR, England, U.K.
Phone: +44 +1223 334476, FAX: +44 +1223 334678
E-Mail: Burkhard.Stiller@cl.cam.ac.uk

Abstract

The support of sufficient Quality-of-Service (QoS) for applications residing in a distributed environment and running on top of high performance networks is a demanding issue. Currently, the areas to provide this support adequately include communication protocols, operating systems support, and offered network services. A configurable approach of communication protocols offers the needed protocol flexibility to react accordingly on various different requirements.

Communication protocols and operating systems have to be parametrized using internal configuration parameters, such as window sizes, retry counters, or scheduling mechanisms, that rely closely on requested application-oriented or network-dependent QoS, such as bandwidth or delay. Moreover, these internal parameters have to be recalculated from time to time due to network changes (such as congestion or line break-down) or due to application-specific alterations (such as enhanced bandwidth requirements or increased reliability) to adjust a temporary or semi-permanent "out-of-tune" service behavior.

Therefore, a rule-based evaluation and QoS updating framework for configuration parameters in a networking environment has been developed. The resulting "rulework" can be used within highly dynamic environments in a communication subsystem that offers the possibility to specify for every QoS parameter both a bounding interval of values and an average value. As an example, the framework has been integrated in the Function-based Communication Subsystem (F-CSS). Especially, an enhanced application service interface is offered, allowing for the specification of various QoS-parameters that are used to configure a sufficient application-tailored communication protocol.

*The author is currently on leave from Universität Karlsruhe, Institut für Telematik, D-76128 Karlsruhe, Germany, and is sponsored by the Commission of the European Communities as a Research Fellow under the Human Capital and Mobility Scheme (RG 19327).

Contents

1 Introduction and Motivation **1**

2 Related Work and Taxonomy **2**
- 2.1 Service Interfaces 2
- 2.2 Service and Resource Management 2
- 2.3 Operating System Support 3
- 2.4 Applied Terminology 3

3 Design of the QoS-driven Update Framework **4**
- 3.1 Discussion of the Problem and Solution Approaches 4
- 3.2 Definitions of CF Parameters 5
- 3.3 Definitions and Examples of Rules 6
- 3.4 Principal Considerations 7
- 3.5 Internal Design of Agents 9
- 3.6 Design of the Rulework 10
- 3.7 Examples 11

4 Implementation and Performance **13**
- 4.1 Implementation Issues 14
- 4.2 Performance Evaluations 15

5 Conclusions **16**

List of Figures

1 Information (1), (2), and (3) Contained in the QoS Deviation Vector 8
2 Principle of Updating CF Parameter Values (Closed-Loop System) 9
3 The Structure of an Agent 10
4 The Rulework in Detail 10
5 The Implementation Structure of the Framework 14

List of Tables

1 Examples of Configuration Parameters 6
2 Examples of Application-requested QoS Parameters, Measured Values, and the Derived QoS Deviation Vector 12
3 Test Requests for an Example Scenario 13
4 CF Parameter Values and Agent's Responsibilities for an Example Scenario . 13
5 Performance Results of the Example Scenario 16

1 Introduction and Motivation

As the variety of application requirements — often expressed in terms of Quality-of-Service (QoS) parameters — increases, the need for efficient end-system architectures becomes clear, since QoS requirements have to be supported efficiently. Traditional end-system architectures followed well-defined models (such as the ISO/OSI Basic Reference Model [1] or the Department of Defense Model [2]) and offered quite static services. Furthermore, corresponding service interfaces included only a very limited set of QoS parameters, which are not extensive, compared to currently required services. As an example, an isochronous service (such as audio or video) may be described by a delay jitter parameter, which is missing from the service specification of the Transport Protocol Number 4 (TP4) [3], [4], while the Transmission Control Protocol (TCP) [5] does not support specific QoS parameters at all.

Traditional protocols are not well suited for appropriate use on gigabit networks [6]. Since established applications did not require various different services, former end-system architectures did and still do not offer the required *service flexibilities*, necessary for modern applications (such as tele-conferencing, tele-learning, virtual reality, or in general multimedia applications) and supported by configurable communication protocols. A real-time video application requires different protocol functionality, *e.g.,* jitter control and synchronization, than a reliable file-transfer, *e.g.,* acknowledgements and checksumming, besides common functionality for both. Therefore, a suitable configuration of a communication protocol can be determined by QoS parameters specified by an application [7]. However in general, other areas of QoS-oriented work (such as modern protocols [8], new architectures [9], enhanced service interfaces [7], [10], and operating system support [11], [12]), which are taken up within the next Section 2, have to be regarded in an integrated manner, providing a suitable solution to QoS guarantees within a networking environment.

Supposing that these approaches solve the lack of service flexibility (an "off-line" problem) and offer solutions for guaranteeing QoS in the end-system and network, the "on-line" situation of adapting configuration parameters sufficiently according to newly arising environmental behaviors still remains open. That is in particular the adaptaion of services during run-time of a communication protocol. In fact, any of these alterations can be made explicitly visible (*e.g.,* by newly issued application requests) or are implicitly detected within the end-system by a monitor, which monitors end-system visible states of the network, such as a link congestion — resulting in a dropping of throughput and increased delay — or increased bit error rates on the links. An appropriate reaction to this detrimental behavior and specific solution for some cases is to keep the requested level of QoS parameter values in the end-system by applying a QoS-driven update of parameters. These *configuration (CF) parameters* are an inherent constituent of communication protocols and stimulate the increase or decrease of certain transport-related or network-dependent QoS parameter values, finally, adjusting the communication subsystem's behavior according to initial application-requested QoS.

The remainder of this work is organized as follows. Section 2 introduces into related work and provides a brief discussion of the taxonomy applied. A clear definition of the "on-line" update problem, its prerequisites and consequences, the design of the architectural framework, and an example are elaborated in Section 3. The prototype implementation of the rulework and its performance evaluations are presented in Section 4 afterwards. Finally, conclusions are drawn in Section 5.

2 Related Work and Taxonomy

Certain environmental issues are important to define an appropriate taxonomy for this framework. On the one hand, applications have to specify their *requirements* to request a special communication service. On the other hand, *network features and services* have to be characterized to be of any use for applications. Therefore, QoS — either for application, protocol, or network features — is expressed within a set of *QoS parameters*.[1] This leads to the discussion of related work on service interfaces before the taxonomy is extended.

2.1 Service Interfaces

In communication protocols or telecommunication systems QoS parameters differ heavily. In the ATM environment parameters such as "cell delay variation", "cell loss rate", or "peak cell rate" have been defined [14], [15]. Transport protocols use different parameters, *e.g.,* "throughput", "delay", "residual error rate", or "priority" in TP 4 [3], [4].

In the Function-based Communication Subsystem [7] the definition of quantitative (*e.g.,* "jitter", "data loss", "data replication") and qualitative (*e.g.,* "ordered delivery", or "intra-stream synchronization") QoS parameters has been proposed. Three different types of QoS values per quantitative QoS parameter are specified. The *threshold value* defines a mandatory requirement for a QoS parameter with the semantics related to that specific parameter (such as minimal needed throughput or maximal tolerable delay). The second value applies to the specific parameter over an amount of time, defining an *average value*. Finally, the *useful value* depicts a limit that bounds the usefulness of the QoS parameter for a specific application (such as minimal usable delay or maximum processable throughput). Each of these values may be utilized for different QoS enforcement strategies [16], [17].

Within the OSI'95 project [10] new QoS parameter definitions are included, *e.g.,* for "transit delay", "transit delay jitter", and "throughput". Furthermore, two types of QoS negotiations are proposed. Various types of services, such as "best-effort" and "guaranteed", are extended by the definition of a "compulsory value" for QoS parameters to allow for the applicability of QoS enforcement strategies. A compulsory QoS parameter value is regarded as a value that has to be monitored and if the limit — negotiated in advance — is exceeded, the requested service has to be aborted. Further approaches comprise a system model providing an application programming interface [18]. A transport system including protocols, resource reservation schemes, and scheduling approaches, has been developed, *e.g.,* in [19]. Finally, the multimedia communication system BERKOM contains an application-oriented service interface [20].

2.2 Service and Resource Management

A flowing transition into the area of service management can be observed, since certain QoS architectures and QoS management schemes inherently rely on well-defined service interfaces and QoS parameters. Therefore, still ongoing work can be found in, *e.g.,* [21], defining a QoS architecture (QoS structures and QoS mapping on scheduling schemes) or as a QoS Management approach in [22]. Furthermore, QoS management has been dealt with in networked

[1] To be more precise, network characteristics are summarized in a set of so-called *network performance parameters* [13], that are in terms of the parameter–value concept not distinctive from QoS parameters except for their area of application.

multimedia systems [23], where features of QoS negotiation, translation, and subsequently resource management are discussed. In [24] a QoS broker model is proposed that allows for the negotiation of QoS parameter values. Additionally, mapping functions between at least the application and the communication subsystem are important to allow for a sufficient and efficient support of applications within certain networks and end-system architectures. Therefore, mapping, enforcement, and monitoring of QoS parameters are considered as an important issue of todays research, but they are not the focus of this document.

2.3 Operating System Support

Additionally, the support of sufficient networking performance relies, as mentioned above, on the operating system as well. An excellent overview of various projects and scheduling mechanisms may be found in [25]. Especially, the scheduling schemes applied to networking tasks are crucial for guaranteeing QoS parameter values. In addition, resource models are developed to allow for the description of schedulable resources that have to be shared or exclusively used by different users. One approach has been developed to integrate scheduling mechanisms, resource administration, and QoS parameter mapping [12]. Further operating system support is done for continuous media in a real-time environment [11] and within the PEGASUS project, to provide a kernel that allows for guarantees of processing and scheduling times for multimedia application streams [26]. However, the focus of the work considered here is not on the operating system in particular, but takes conceptually into account a possible parametrization of scheduling mechanisms within the operating system. These parameters can be regarded as system resources (cf. Subsection 2.4) that may be updated as necessary.

2.4 Applied Terminology

The term of *resources* is applied in the presented framework of QoS-driven updates. Resources cover three distinctive areas determining a number of *related QoS parameters*. Firstly, *network resources* are used to describe features and characteristics of networks that are connected via certain network adapters to an end-system. Relevant parameters for their characterization vary according to the network. Some examples include features, such as "broadcast support", "bit corruption rate", or "packet loss rate". Network performance parameters, as defined in I.350 [13], may be applied as well. Secondly, *system resources* define features of the end-system itself, such as "CPU performance", "net interface bandwidth", "memory", "buffer size", or "scheduling strategy", including operating system aspects. Thirdly, *protocol resources* describe atomic building blocks that are utilized to configure a communication protocol [27]. They are hierarchically structured as *protocol functions* — examples include "acknowledgement", "checksumming", or "flow control" — and *protocol mechanisms*, such as a "selective acknowledgement" or a "cumulative acknowledgement", a "window-based flow control" or a "rate-based flow control". Concerning the use of the term *QoS parameter*, various slightly different definitions exist. Therefore, in the following it is regarded in particular as a generic term for network and system resource parameters, protocol-related internal configuration parameters, and application-oriented QoS parameters. If necessary, refinements are described elsewhere in the document.

Communication protocols or, more accurate, their atomic building blocks in terms of protocol functions and mechanisms, influence QoS parameters, network resources, and system resources. For example the window-based flow control — if it works correctly — is dependent on the number of available buffers in an end-system; it tries to prevent packet losses within the network; and it has to be parametrized internally by a window-size parameter. In general, the utilization of protocol resources stimulates directly or secondhand the increase or decrease of certain QoS parameter values as such. Additionally, protocol resources may be defined internally by *configuration (CF) parameters*, such as the "window size" of the window-based flow control mechanism or the "retry counter" of an acknowledgement function. These CF parameters do have a large effect on the process of decreasing or increasing QoS parameter values. Therefore, CF parameter updates according to certain QoS parameter values in the specific situation may lead to a sufficient support of application-requested QoS.

Finally, an arithmetical or logical expression, consisting of QoS parameters, may form a *rule* that specifies the linkage between parameters and possible impacts on them. *E.g.,* an increase of packet errors in an end-system leads to a drop of the application-usable bandwidth.

Local states within an end-system and global ones of the connected network are monitored by a *monitor*. This tool is responsible for detecting changes and variations in the current situation, *e.g.,* load, throughput, delay, jitter, or error rate, and it is responsible for keeping relevant values in a data base for further investigations, such as for the QoS-driven update of CF parameters or employing them to various enforcement strategies.

3 Design of the QoS-driven Update Framework

The main goals for supporting communication needs between users or end-systems within a distributed environment are a sufficient, flexible, and adaptable framework including communication protocols, operating system, and network issues. One aspect covers the adaptivity. This will be extended within this document on updating configuration and QoS parameters of communication protocols. Besides, operating system parameters (such as for scheduling mechanisms or memory management schemes) are considered. This becomes especially relevant in a distributed environment, *e.g.,* where multiple users participate in a globally distributed teleconferencing scenario and the network performance degrades from time to time for various reasons. As pointed out in the related work section, ongoing work for additional scenario evaluations of an integrated handling of relevant QoS parameters occurring in an end-system are important and for further study. Therefore, this section deals with an experimental approach of maintaining CF and QoS parameters and their updates in a flexible protocol configuration environment.

3.1 Discussion of the Problem and Solution Approaches

A communication protocol is used to transfer data between users or applications residing on top of end-systems. If service requirements of an application can not be met any more by (1) the currently applied protocol, (2) the underlying communication subsystem, and/or (3) the network[2], in principle seven different choices of handling the situation are possible:

1. The values of initially application-requested QoS parameters will be changed.
2. The primary values of CF parameters will be updated without altering application-requested QoS, but stimulating changes of transport-related QoS for getting adjusted to the application-requested QoS. The protocol's functionality remains unchanged.

[2] This may be detected by a monitor, after a comparison of measured values with certain bounding values has been carried out.

3. The protocol will be reconfigured, while changing the protocol's functionality, *e.g.,* taking functions in or out, for getting adjusted to the application-requested QoS.
4. The entire communication protocol in use will be exchanged with a different one to provide the application-requested QoS.
5. The end-system may offer a certain degree of QoS, which will be only available at this stage, and the application accepts this proposal, while using the old protocol onwards.
6. The data transfer will be kept as before without any changes of the protocol, but with a changed service.
7. The data transfer will be aborted.

Solution 1 suffers from the problem that application-specific QoS have been set initially according to certain application demands and, therefore, alterations are not helpful. They result in the change of application requirements instead of changes within the supporting system, if feasible at all. For that reason, simple changes in the protocol should be taken into account first, such as in solution 2. Any protocol related CF parameter may be changed, if a number of previously defined rules is abided by. *E.g.,* the window-size of a flow control mechanism may be increased, if sufficient memory is available and a constant throughput has to be maintained. This update stimulates the change of transport-related QoS, which in turn adjusts the overall behavior to the application-requested QoS if possible. If changes of these CF parameters do not achieve a proper behavior of the overall system, solution 3 may solve the problem, if (re-)configuration tools are available [27], [28], [29], such as encountering a retransmission function, if the requested reliability drops below a defined limit. Otherwise solution 4 can be regarded as a coarse-grained reconfiguration task, *e.g.,* exchanging TCP for UDP. Solution 5 deals with a rather end-system-oriented view, which proposes service only that currently available [30]. Finally, if neither previously discussed solution succeeds, the service of the protocol and the data transfer may be kept as before (solution 6) or will be completely aborted (solution 7). Obviously, this decision depends on tolerance features of the considered application.

As solution 1, 6, and 7 can be implemented quite easily and solution 3, 4, and 5 are discussed elsewhere [27], [28], [26], [29], the remainder is focussed on the pertinent framework to solution 2, the update of CF parameters. This approach is reasonable in various cases, where the service offered to the application has to be changed, but reconfigurations of the protocol's functionality are considered as too complex.

3.2 Definitions of CF Parameters

First of all, several examples of *configuration (CF) parameters* are listed in Table 1 to motivate their beneficial definition. CF parameters are collected from various, *e.g.,* transport protocols, and each of them is defined by three components: a *unique identifier*, its *type of value*, and its *unit* of observation. The considered protocol resource (column 1 of Table 1), in particular a protocol mechanism (cf. Subsection 2.4) belongs to a certain protocol function (depicted in italics). Each protocol mechanism has an unlimited, but fixed number of CF parameters.[3]

The identifier of a CF parameter is a specification as an ASCII-readable string format, such as `SR_WS` or `CRC_POLYNOM`. The type of value may belong to one of the following categories:

[3] If a protocol function includes a CF parameter that is similar for every possible mechanism of the considered function, it is listed in the corresponding table's line of the function's name. *E.g.,* every single mechanism for the function "acknowledgement" includes the CF parameter `ACK_RETRY` besides the mechanism-specific ones.

Protocol Resource	CF Parameter	CF Identifier	Type of Value	Unit	Effect
Retransmission					
Go-back-N	Window size	`RTM_GBN_WS`	exp-2	Byte	Memory
Selective repeat	Window size	`RTM_SR_WS`	exp-2	Byte	Memory
Forward error correction 1	Redundancy factor	`FEC1_RED_FAC`	nat[0..5]	-	Memory
Forward error correction 2	Redundancy factor	`FEC2_RED_FAC`	nat[6..999]	-	CPU
Checksumming					
Cyclic redundancy check	Polynom	`CRC_POLYNOM`	nat	-	CPU
	Range	`CRC_SCOPE`	disc	-	CPU
Flow control					
Window-based	Window size	`FC_WS`	nat	Byte	Memory
Rate-based	Inter-packet distance	`FC_IPD`	cont	ms	CPU
	Timer-driven counter	`FC_COUNTER`	cont	-	CPU
Congestion control					
Reactive	Max window size	`CC_MWS`	exp-2	Byte	Memory
	Retry interval	`CC_RETRY`	nat	-	CPU
	Sending rate	`CC_SRATE`	cont	Byte	Memory
Preventive	Increase of load	`CC_INC_LA`	cont[0..1]	-	CPU
Acknowledgement	Retry counter	`ACK_RETRY`	nat	-	CPU
Packet-based selective	Number of packets	`ACK_S_NP`	nat	PDU	CPU
Packet-based cumulative	Number of packets	`ACK_C_NP`	nat	PDU	Memory
Byte-based cumulative	Number of bytes	`ACK_C_NB`	nat	Byte	Memory
Timer-based	Timeout	`ACK_TIME`	cont	ms	CPU

Table 1: Examples of Configuration Parameters

- natural (nat)[4],
- integer (int),
- continuous (cont)[5],
- exponential to the power of y (exp-y), while $y \in$ nat,
- boolean (bool)[6], where two discrete values exist, or
- discrete (disc), where multiple discrete values exist.

Any numerical value of the types "nat", "int", "cont", or "exp-x" may be additionally bounded by certain emphasized values. For that reason, an interval definition, such as "[0..1]" may be specified. Values of the "disc" type have to be specified by enumerating the appropriate set of values. Relevant units per CF parameter, such as milliseconds (ms), bytes, protocol data units (PDU), or none (-) are added in a separate column. Further explanations are listed in the table, identifying CF parameters' principal effects on system resources, which encompass the *CPU* (Central Processing Unit) performance for processing protocol-relevant information and the *memory* needed to store data units or intermediate results.

3.3 Definitions and Examples of Rules

The semantics concerned with a single QoS parameter or CF parameter have to be well-defined to be usefull on their own. Additionally, various linkages between these parameters define impacts on themselves or others. As an example the increase of the redundancy factor leads to higher throughput or a highly initialized retry counter may lead to lower user accessible

[4] The type "nat" is a special case of "int": $\forall$ x $\in$ nat: x $\geq 0 \wedge$ x $\in$ int.
[5] The type "cont" encompasses the positive part of real numbers: $\forall$ x $\in$ cont: x > 0
[6] The type "bool" is a special case of "disc".

throughput in an unreliable environment. Furthermore, the importance of these impacts can be distinguished as a certain weight. Any of these linkages that can be expressed in form of a precondition (**IF**), an impact (**THEN**), and a weight (**WEIGHT**), is called a *rule*.

IF `A = <value_1>` **THEN** `B := <value_2>` **WEIGHT** `<value_3>`

In any case, where the contents of the QoS or CF parameter identifier `A` equals some type of `value_1`, another identifier `B` will be assigned to `value_2` and the result is weighted by `value_3`. In a more general form, any arithmetical or logical expression [7] can be used to specify a precondition or an impact, even consisting of multiple assignments. Therefore, the following example encompasses the above stated one in a formal manner:

IF `FEC1_RED_FAC = 2` **THEN** `THRPUT := THRPUT * 1.5` **WEIGHT** `0.9`

It is a major task of the system's designer to identify valid and important rules within an arbitrary, but fixed environment. Most of the rules are based on observations of the communication system, others are the result of logical derivations. Additionally, a rule shall be made as simple as possible, which is in particular only a single precondition and a single impact, presenting some comparison of a parameter value with a certain bound and its impacts on one or two others. Additionally, the focus of a rule should be clear in terms of its main intention, which is represented by assigned weights to define an impact according to a special CF and QoS parameter or system resource. For example, one main implication of a rule is an extensive impact on the user-applicable throughput (*e.g.,* weighted 1.0) and a minimal impact on delay issues (*e.g.,* weighted 0.1). Complicated rules may be used as well, but the applicability of these rules to more general scenarios might be negatively affected. Any contradictions or incompleteness between rules will be dealt with within the rulework (cf. Subsection 3.6) as far as useful and possible. Finally, a single rule will be an inherent part of a special agent (cf. Subsection 3.5).

3.4 Principal Considerations

Depending on the fact that only certain CF parameter values may be changed and that the update has to take place within certain well-defined circumstances, the process of selecting an updated CF parameter value relies on three different factors:

1. **Proportion of Difference** — The proportion of difference for a specific QoS parameter value is defined as the difference between an originally specified QoS parameter value and the measured value. Any deviation is depicted on a per QoS parameter basis in a vectorized manner (*QoS Deviation Vector*) presenting the deviation of the average and threshold/useful value.[8] Therefore, each QoS deviation vector contains the following information (cf. Figure 1):

   (1) the measured value V_m of the QoS parameter in absolute numbers,

   (2) the deviation δ_{lim} of V_m from the threshold value V_t or the useful value V_u — depending on the closest distance — in a relative percentage ($\pm$ x%), and

   (3) the deviation δ_{opt} of V_m from the average value V_a in a relative percentage ($\pm$ x%).

[7] The developer has to take care of a correct handling of the units of observation for each applied parameter, since a multiplication of a disc-type parameter with an int-type parameter is not defined.

[8] Depending on the measured value, either the threshold or the useful value may have fallen short of/exceeded. Therefore, only one single deviation value is useful as specified within the QoS deviation vector.

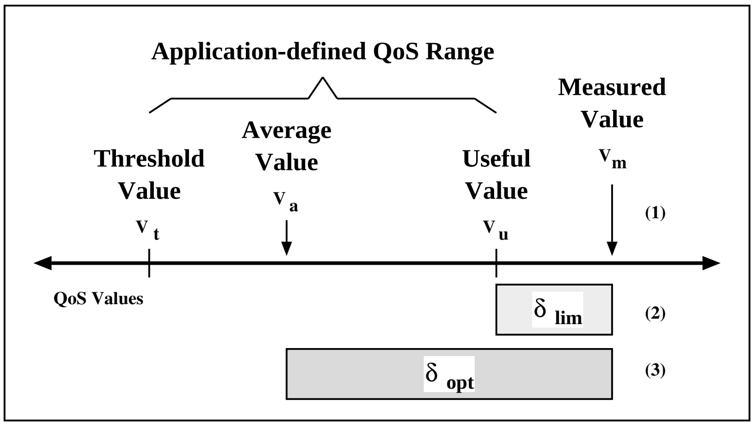

Figure 1: Information (1), (2), and (3) Contained in the QoS Deviation Vector

The average value of a QoS parameter acts as the target value for the update process. Concerning agents (cf. Subsection 3.5), percentages are very helpful for defining relative distances between measured values and targeted numbers. The absolute number of V_m will be used for final decisions in agents. An example of a QoS deviation vector can be found in Subsection 3.7.

2. **Set of Considered QoS Parameter** — Since some protocol mechanisms are dependent on others, the relevant QoS parameters are dependent as well. Additionally, QoS parameters are considered for the update process only, if increasing or decreasing effects of them can be achieved within the currently configured communication protocol. Therefore, certain CF parameters as defined in Table 1 may not be relevant as well and do not need to be updated.

3. **Current Situation of Resources** — If an update of a CF parameter value occurs, the updated demand on system resources have to be checked. Therefore, each CF parameter specification contains an additional arithmetical expression that defines dependencies on system resources. The expression — called CF parameter specific information — determines, for example the increase or decrease of memory or CPU performance.

The Closed-Loop System for Calculating Updates — Figure 2 depicts the entire principle of updating CF parameter values. Application-requested QoS parameter values and measured QoS parameter values are used to calculate the QoS deviation vector. This vector in addition to system resource information is fed into the *rulework*, which is defined internally by multiple *agents*. The rulework is responsible for calculating the updated values for relevant CF parameters, which are currently used in the communication protocol. Besides system resource specifications, the resource data base includes for each CF parameter the currently valid value as well. The monitor monitors application-specific, network-dependent, and protocol-related QoS parameter values and issues, if violation conditions apply, newly measured QoS parameter values.

As it can be seen, a closed-loop system is necessary, instead of an open-loop system to allow for feedback signals taken from protocols. Obviously, the decision "when to start the described process again" effects the stability of the protocol's behavior. The tradeoff between timeliness and stability is for further detailed study, while keeping in mind that certain boundaries and thresholds for parameters will be considered with high priority, before

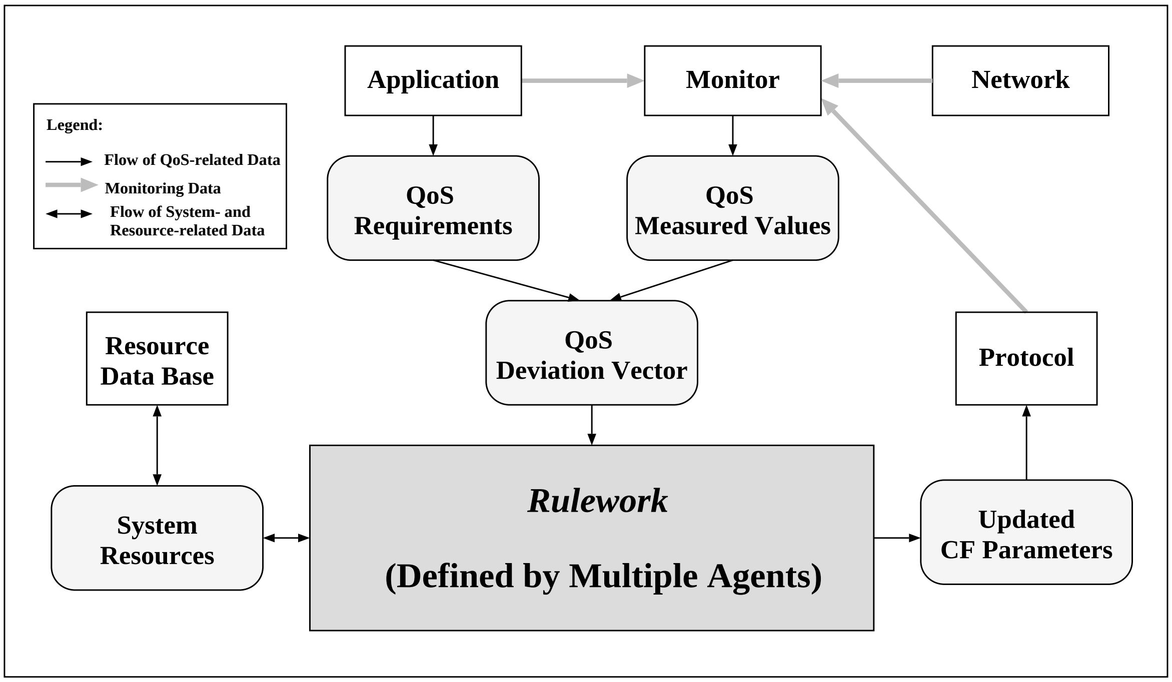

Figure 2: Principle of Updating CF Parameter Values (Closed-Loop System)

any update procedure is due to start. The exploitation of QoS guidelines as proposed within the QoS basic framework is intended [16].

3.5 Internal Design of Agents

The calculation of a proposed CF parameter value update is done by an *agent*. Based on an agent's specific subset of CF parameters Ω' of the entire set of CF parameters available Ω and on the QoS deviation vector Δ, a weighted proposal Σ of CF parameters $\omega \in \Omega'$ has to be calculated. Therefore, every agent consists of three different components:

- **Input Check** ζ — This input check decides, whether the agent is activated and if all agent specific CF parameters form a subset of the entirely available and to be updated CF parameters: $\Omega' \subset \Omega$.
- **Processing Unit** ϕ — The processing unit calculates for each CF parameter ω in the subset Ω' ($\omega \in \Omega'$) a separate CF parameter value update proposal.
- **Weighting Function** Σ — The weighting function weights each calculated CF parameter update proposal V_{upd} resulting in the agent's output.

The internal behavior of the processing unit is statically pre-defined at definition time of the agent, while implementing exactly one defined rule. Agents may be specified independently of another and may operate on an arbitrary, but fixed number of CF parameters. The interaction of multiple agents is defined within the rule-based framework using the filter as defined in the next Subsection.

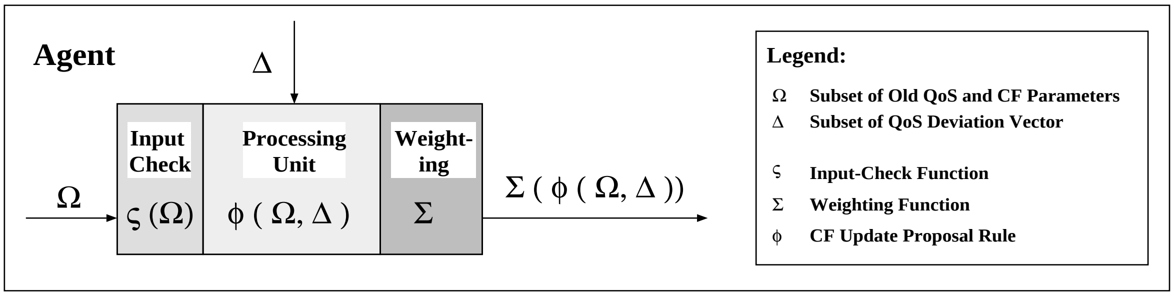

Figure 3: The Structure of an Agent

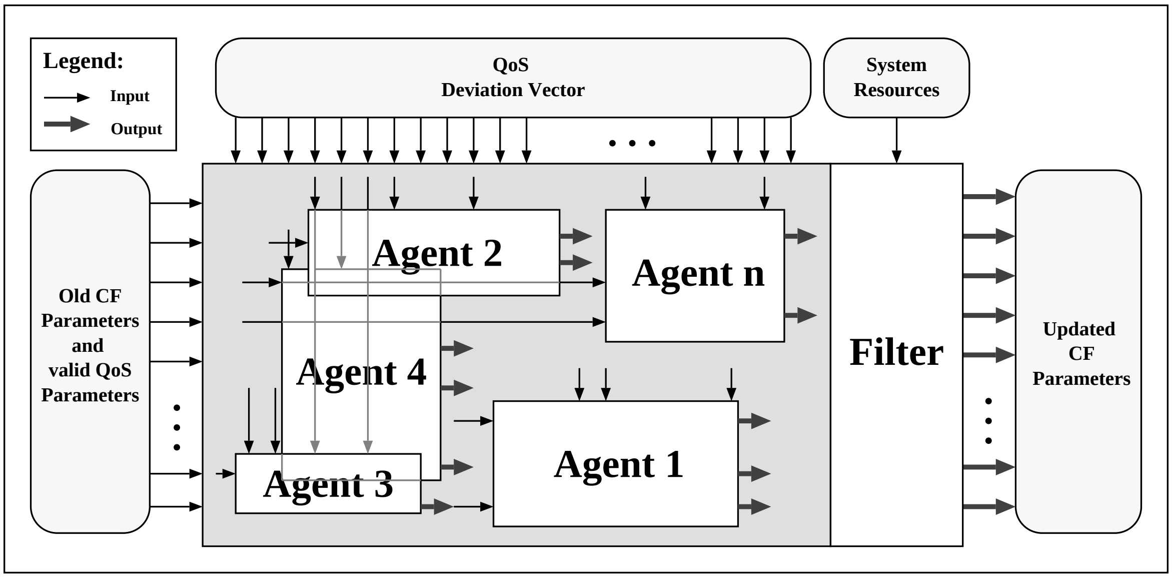

Figure 4: The Rulework in Detail

3.6 Design of the Rulework

The basic element of the rulework is an *agent*. Generally, each agent handles a certain subset of CF and QoS parameters, its deviations, and its interdependencies (agent input). As it has been explained, the result of an agent (agent output) is a weighted proposal of one or multiple CF parameter value updates, while applying the agent's internal rules. Figure 4 presents the designed rulework, which calculates, due to the currently valid QoS derivation vector, for each regarded CF parameter the resulting CF parameter value or detects incompatibilities and contradictions.

An important precondition is the fact that any weighted CF parameter value update (an agent's output) has to be a subset of the currently regarded CF parameters (input to all available agents) and being part of the communication protocol itself. Also important to recognize is a possible overlapping of input and output sets of CF parameter values for multiple agents. Since the rules are defined from a single agent's perspective only, contradictions in the separately proposed updates of CF parameter values by multiple agents are likely to occur.

Therefore, the *filter* is responsible for synthesizing all separate outputs of active agents[9] to one single final result. In this case three different steps of the filter have to be processed for every available CF parameter z :

1. Check for normalized weights:
   (a) Calculate with all n available CF parameter value update proposals V_{upd} and the corresponding CF parameter weights of the agents δ a normalized weight $\delta' = 1$ based on all n proposal's weights.
   (b) Apply the normalized weight δ' to calculate the average weight $\delta'(z)$ for each CF parameter z separately.
   (c) Check for any contradictions between possibly different proposals for the same CF parameter.[10]
2. Check for necessary system resources.
3. Check for admissibility of the updated and finally average weighted proposal examining types of value and bounding intervals.

For each step defined above, certain conditions to reject a proposed value have been defined, which is being detected by the filter. The reject conditions encompass:

1. Highly weighted, but contradictory agent outputs for a CF parameter z indicate a low stabilized result, since a normalized average weight $\delta'(z)$ would result in a value next to zero and prevents any intended change of the CF parameter z value.
2. A request for additional system resources could not be met.
3. The calculated resulting CF parameter value is placed out of valid intervals or does not belong to the correct type of value.

In each case of rejection, the initially requested QoS parameter values of the application can not be guaranteed within the current configuration (protocol functionality and CF parameters as well). Therefore, the update of CF parameters is not possible and a reconfiguration of the complete communication protocol may be stimulated (cf. Subsection 3.1, 3rd item).

3.7 Examples

In this Subsection several examples of existing CF parameters, the calculation of a QoS deviation vector, and of prototyped agents are presented. A quite huge number of CF and QoS parameters is necessary to implement a realistic scenario including valid parameters and values. Therefore, the important prerequisites and steps are discussed, but simple arithmetical operations, such as averaging or weighting, are omitted for simplicity. The following presentation with tables and explaining text appears advisable to clearly divide numerical facts from commentary notes.

[9] The conditions for an activation of agents are examined by the input check (cf. Subsection 3.5.

[10] Contradictions occur, if at least two CF parameter value update proposals for a single CF parameter show opposite signs (+/-). Proposals are considered as neutral, if the value update proposal equals zero.

QoS Parameter	Unit	Threshold	Average	Useful	Measured	δ_{opt}	δ_{lim}
Throughput	[Mbit/s]	600	800	1000	570	–28.75 %	–5.0 %
Delay	[ms]	40	30	20	40	+46.67 %	+10.0 %
Jitter	[ms]	7.5	10.0	12.5	15	+50.00 %	+20.0 %
Bit Error Rate	$[-log_{10}]$	10	15	20	14	–6.67 %	±0.0 %

Table 2: Examples of Application-requested QoS Parameters, Measured Values, and the Derived QoS Deviation Vector

CF Parameters — The rate-based flow control mechanism includes two CF parameters (cf. Table 1). Both of them are marked by a "cont" type of value, where the unit for the inter-packet distance (`FC_IPD`) is defined as milliseconds and the time-driven counter has no unit. Their main influences on system resources effect the CPU performance of the end-system. Another interesting CF parameter is `CRC_SCOPE`. The type of value is "disc", which is specified as a set of three discrete values: {`Header, Data, both`}. In turn, they define the scope of the "cyclic redundancy check" mechanism to be used for a protocol data unit.

QoS Deviation Vector Calculation — Table 2 depicts a short example for the calculation of a QoS deviation vector of the dimension 4. An application defined the QoS parameters throughput, delay, jitter, and bit error rate according to column 2, 3, and 4, while the monitor allowed for the measurement of the current valid values according to column 5. The calculated QoS deviation vector dimensions δ_{opt} and δ_{lim} for these numbers are included in columns 6 and 7, additionally, according to Subsection 3.4, 1st item. δ_{lim} for the bit error rate has been set to ±0.0, because the measured value of 14 $[-log_{10}]$ is located within the permitted bit error rate's interval of 10 and 20 $[-log_{10}]$ and, therefore, no violation of these limits has occured.

Preset CF Values and Agents — An example for two different QoS deviation vectors of dimension 4 and 5 — four and five QoS parameters are considered, respectively — and two different measured values (M) for request 1 (R1) and request 2 (R2) is presented in columns 1 to 7 in Table 3. Additionally, the preset values of ten relevant CF parameters are specified in two different ways in columns 2 and 3 of Table 4. These CF parameters are considered exemplarily for a transport-related communication protocol that offers a non real-time and reliable type of service, since the protocol functions acknowledgement, flow control, and checksumming are considered. Further operating system-related resources may be utilized as well, but do not form a part of this, already quite complex example.

Furthermore, prototyped agents A1, A2, and A3 operate on certain CF parameters (depicted in column 4, 5, and 6 of Table 4 by an "×"). These agents consider certain QoS parameters differently weighted as *important* (weight 1.0), as *of interest* (weight 0.5), or as *not important* (weight 0.0) marked as "–" (cf. columns 8, 9, and 10 in Table 3), depicting the main focus of the specified rule within an agent. [11] For example, agent A1 focusses mainly on the bit error rate, since it is responsible for the CF parameters of the acknowledgement and checksumming mechanisms. Therefore, the impact on the bit error rate is weighted by "important". However, the processing of acknowledgement and checksumming do have smaller impacts on the throughput and the delay of data units. Therefore, these impacts are marked

[11] For simplicity reasons only these three categories of weights have been used within this example. In general, all different weights in the numerical interval of 0.0 and 1.0 may be utilized.

QoS Parameter	M (R1)	δ_{opt} (R1)	δ_{lim} (R1)	M (R2)	δ_{opt} (R2)	δ_{lim} (R2)	Agent 1 (A1)	Agent 2 (A2)	Agent 3 (A3)
Throughput	500	-10%	0%	670	-3%	0%	of interest	–	–
Delay	105	+30%	+1%	25	-5%	0%	of interest	important	–
Jitter	15	-65%	-1%	85	+80%	+20%	–	important	–
Bit Error Rate	6	-80%	-10%	12	+20%	0%	important	of interest	important
Bit Loss Rate	–	–	–	10	-1%	0%	–	–	important

Table 3: Test Requests for an Example Scenario

CF Identifier	Preset CF Value (R1)	Preset CF Value (R2)	Agent 1 (A1)	Agent 2 (A2)	Agent 3 (A3)	Proposed Update (R2)
`ACK_C_NP`	16	2	×	×	–	3
`ACK_TIME`	0.5	0.1	×	×	×	0.13
`ACK_RETRY`	3	1	×	×	×	2
`FC_WS`	32	32	–	–	–	32
`CC_SRATE`	0.25	0.25	–	×	–	0.24
`CC_MWS`	16	32	–	×	–	16
`CRC_SCOPE`	2.00	1.00	×	–	–	1
`CRC_POLYNOM`	16	16	×	–	–	16
`FEC1_RED_FAC`	–	2	–	–	×	1

Table 4: CF Parameter Values and Agent's Responsibilities for an Example Scenario

by the weight "of interest". As it can be seen, the specification of impacts and weights is quite tricky, but not impossible. A careful analysis of impacts and consequences of either rule leads to a usable data base for agents. In turn, agent A2 focusses on delay and jitter issues (weighted as "important"), while maintaining acknowledgement and congestion control parameters, mainly. The bit error rate is considered as "of interest", since acknowledgement mechanisms — and the retransmission of course — allow for an improvement. Finally, agent A3 focusses on the bit error and loss rate, since CF parameters of the time-driven acknowledgement mechanism and the forward error correction scheme are maintained.

For the scenario R2 all three agents and for R1 agents A1 and A2 will be activated, since for R1 no bit loss rate QoS parameter has been specified (cf. Table 3, column 2). Therefore, two CF parameter value update proposals will be generated for `ACK_C_NP`, `ACK_TIME`, and `ACK_RETRY` (cf. Table 4 columns 4 and 5). The CF parameters `FC_WS` and `FEC1_Red_FAC` will not be used for scenario R1, since no agent operates on it. These examples will be used in Subsection 4.2 to evaluate the performance behavior of the rulework.

Finally, applying all steps as defined in previous Subsections 3.5 and 3.6, the proposed update for the CF parameters in scenario R2 is presented in Table 4 column 7. As it can be seen, while examining the units of observation of CF parameters in Table 1, the results match and are sufficient for the considered scenario. The results for scenario R1 are not shown, since a contradiction — the proposed new values for CF parameter `ACK_C_NP` have been calculated to −15 and 16 — occured within the proposed results, which is for that reason abandoned.

4 Implementation and Performance

This Section focusses on internals of the prototype implementation of the rulework/agents and subsequently their prototype performance evaluations. Issues, regarding the communication with the monitor, receiving application QoS, and finally transmitting the updated CF pa-

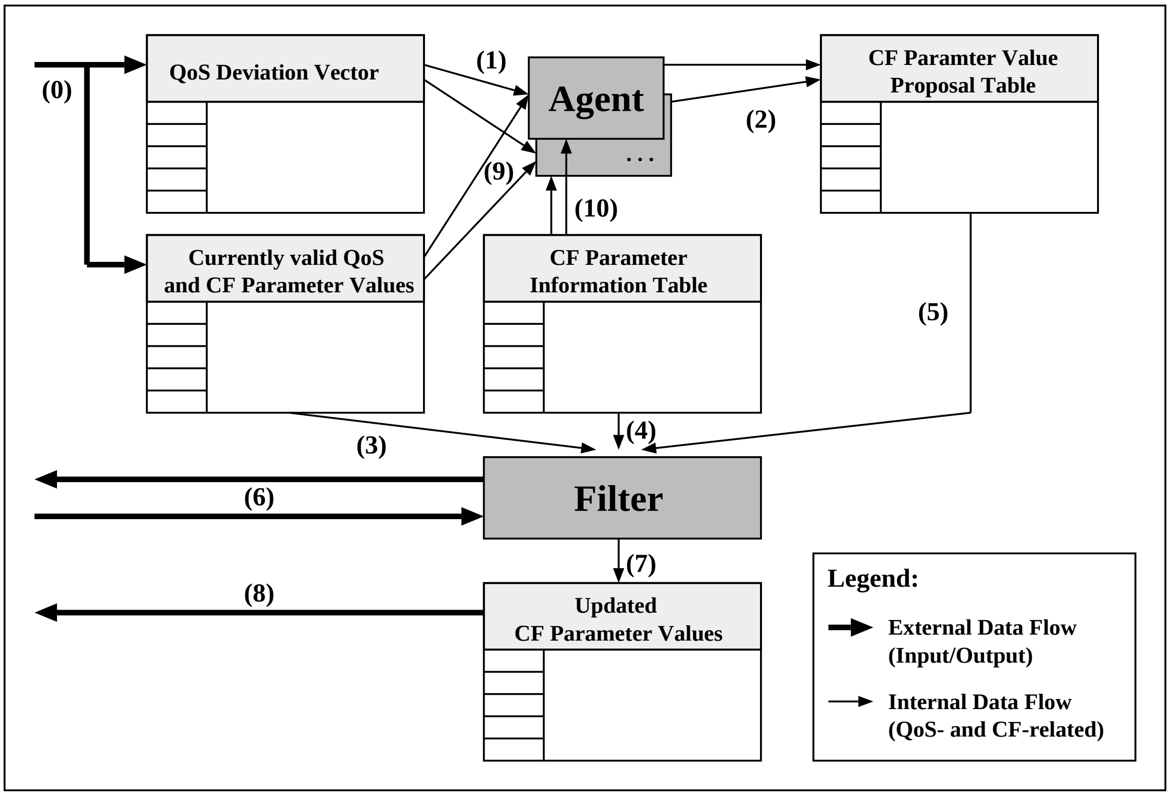

Figure 5: The Implementation Structure of the Framework

rameter values to the protocol function or mechanism may be found in [31]. Processing times of reserving system and network resources, changing scheduling mechanism parameters, or other operating system dependent issues are not taken into account. However, the developed solution of QoS updates provides an experimental framework that allows for the processing of well-defined updates in a networking environment. Additionally, the impacts of the update proposal calculation time on different, already active tasks within the regarded end-system has to be studied. Any type of interference between them has to be avoided according to a guaranteed service behavior of the end-system.

4.1 Implementation Issues

Since the rulework may consist of multiple agents that operate on the same input data, a process concept has been applied. Every agent will run as a subprocess (Transputer-based thread) of the rulework. Furthermore, the agent and framework interfaces are defined and implemented in C++ to allow for an easy addition of supplementary agents into any existing rulework.

The rulework acts as a distributer of incoming QoS and CF parameter values to multiple agents. Afterwards, the results of each agent is used to feed the filter for calculating the final CF parameter values. As depicted in Figure 5, the internal flow of data is similar to the modelling as described in Figure 2. All incoming QoS and CF parameter values (0) are stored in tables and are handed to the agents (1). The agents apply their internal rules and propose a local result. Any proposed value is registered in the "proposal table" (2), while the CF parameter identifier operates as an appropriate table index. Now the filter operates on currently valid parameter values (3), CF parameter specific information (4), *e.g.,* increasing or decreasing effects on system resources, and proposed values (5). Immediately, according to the steps defined in Subsection 3.6, the check for normalized weights is processed, a system resource request will be issued (6), and the admissibility check will be done. Finally, the resulting set of CF parameter value updates will be written into the "update table" (7) and handed out to the communication protocol and the resource data base afterwards (8).

An agent operates as a single thread on a number of QoS and CF parameters. According to their relevance (needed versus not needed), an agent initializes its input with the current values for each required parameter, while a copy of these values into local variables takes place. The processing of these agent rules and the weighting function will follow only, if the input check of the agent succeeded. The prototype implementation of agents integrated the processing unit and the weighting function into one single code segment and their results will be written into the "proposal table". As depicted in Figure 5, the processing unit of an agent has access to the table of current valid QoS and CF parameter values (9) as well as to CF parameter specific information (10).

Finally, a considerable implementation decision of the rulework is explained. Since a potential huge number of QoS deviations may occur, a global CF parameter-based table would need a large number of entries, which is a consequence of the number of possible numerical combinations. Hence, a considered communication protocol does not offer every single QoS parameter, which is generally available. Therefore, only a small subset of combinations of CF parameters are useful in a specific circumstance. Additionally, a fairly huge number of similar entries in the global table would be due to the fact that some combinations of QoS parameters are completely independent of another. Therefore, the mentioned smaller tables as well as the rules on the per-agent bases have been implemented to reduce the complexity of a global, almost empty table. Furthermore, the required flexibility in terms of adding or withdrawing agents in the rulework has been achieved easily. Instead, a global table-based implementation would make that type of handling almost infeasible.

4.2 Performance Evaluations

The prototypical implementation has been evaluated feeding several scenarios into the rulework and some agents. Two example scenarios, as they have been described in Subsection 3.7, are considered for performance measurements. The framework and three agents are activated with data according to Table 3 and 4. First of all, the evaluation of a correct filter behavior has been done, since request 1 (R1) did not allow for an update of CF parameter values and request (R2) proposed certain updated values. The values of the QoS deviation vector and "measured" values[12] are included in Table 3. Finally, Table 4 includes for both requests previously initialized parameter values.

Agents and the filter have been performance evaluated. The resulting processing times of the implementation consist of two distinct portions: static and variable portions. Static portions of time are unavoidable, since they are the result of the framework itself. Variable portions depend on the input check, the processing unit, and the weighting function of agents and on the weight checks of the filter. Table 5 presents the performance evaluated and

[12] These values have been assumed to be measured values, since no running monitor was available at the time of evaluation.

Requests No.: m, p, q	3 Agents [μs] T_a measured	3 Agents [μs] T_a derived	Filter [μs] T_f measured	Filter [μs] T_f derived	Sum [μ] measured	Sum [μ] derived
R1: 2.5, 8, 10	1170	1162	684	5295	1875	6457
R2: 3, 9, 12	1381	1395	5062	6015	6443	7410

Table 5: Performance Results of the Example Scenario

analytically derived performance results according to the example in Subsection 3.7, where m denotes the number of active agents, p the number of considered CF parameters, and q the number of proposed value updates. The analytical model is quite simple (worst case) and assumes that in average the processing times for agents (T_a) are similar. Therefore, the overall derived processing time for the framework ($T_{derived}$) corresponds to the following equation, while m agents are active and T_f depicts the processing time of the filter:

$$T_{derived} = m * T_a + T_f$$

In request R1 agent A3 will not be activated after the input check has been done. Therefore, the processing time for $m = 2.5$ agents is used in the analytical model. Additionally, in this specific situation $p = 8$ CF parameters have been considered and $q = 10$ update proposals have been calculated, where two of them resulted in the above mentioned contradiction. As the numbers for the agents show, the difference between measured and calculated numbers is marginally. Concerning the filter, the differences are quite huge. The reason can be found in the detection of a contradiction between two proposals and its subsequent abort of processing, while the modelling allows for the calculation of the worst case without any knowledge on possible contradictions.

In summary, for request R1 approximately 1.9 ms are necessary to calculate the contradiction, while R2 takes about 6.5 ms for proposing a set of new CF parameter update values (cf. Table 4, column 7). Further evaluated examples show different final sums for processing times, but the range of absolute numbers is quite similar for a given number of alike agents.

5 Conclusions

The discussion of updating CF parameters according to QoS parameter values led to the design of the rule-based framework including independent agents. They form the basic components to calculate CF parameter value update proposals, that will be accumulated in a filter for the final update value. Depending on certain input values — old QoS and CF parameter values as well —, an appropriate result of the rule-based framework in a networking environment is:

(a) a final decision to update certain CF parameter values, while presenting a list of updated CF parameter values, or

(b) initiate an entirely new reconfiguration.

Advantages of the rulework are the flexible architecture, where adding or removing of agents is simple. Additionally, the portability of the entire framework is important, since it does not include any system specific prerequisites. Basic requirements include the possibility to specify intervals and average values for QoS parameters as well as measured values.

Finally, the update decision will be calculated in a reasonable amount of time (less than 10 ms in the examples), as the performance evaluations present. Therefore, the use of an agent-based closed-loop system to evaluate update conditions in a high-performance environment is feasible, except for very short-termed real-time applications.

Acknowledgements
Many thanks go to Wolfgang Janzen, who implemented prototypically the rule-based framework and some agents. Additionally, I am indebted to Kobus van der Merwe, who discussed and proofread previous versions of this document.

References

[1] ISO Standard, IS 7498, *Information processing systems — Open Systems Interconnection — Basic Reference Model*, 1985.

[2] D. Comer, *Internetworking with TCP/IP Vol I: Principles, Protocols, and Architecture, 2nd edition.* Englewood Cliffs, New Jersey, U.S.A.: Prentice Hall, 1991.

[3] ISO Standard, IS 8072, *Information processing systems — Open Systems Interconnection — Transport Service Definition*, 1986.

[4] ISO Standard IS 8073, *Information processing systems — Open Systems Interconnection — Transport Protocol Definition*, 1988.

[5] DARPA, *Transmission Control Protocol — DARPA Internet Protocol Specification, RFC 791*, September 1981.

[6] T. LaPorta and M. Schwartz, "Architectures, Features, and Implementation of High-Speed Transport Protocols," *IEEE Network Magazine*, vol. 5, pp. 14–22, May 1991.

[7] M. Zitterbart, B. Stiller, and A. Tantawy, "A Model for High-Performance Communication Subsystems," *IEEE Journal on Selected Areas in Communication*, vol. 11, pp. 507–518, May 1993.

[8] W. T. Strayer, B. J. Dempsey, and A. C. Weaver, *XTP: The Xpress Transfer Protocol.* Reading, Massachusetts, U.S.A.: Addison Wesley, 1992.

[9] M. Zitterbart, "High-Speed Transport Components," *IEEE Network Magazine*, vol. 5, pp. 54–63, January 1991.

[10] A. Danthine, *The OSI'95 Transport Service with Multimedia Support — Research Reports ESPRIT, Project 5341, Volume 1.* Berlin, Germany: Springer, 1994.

[11] E. A. Hyden, "Operating System Support for Quality-of-Service," Tech. Rep. 94-340, University of Cambridge, Computer Laboratory, Cambridge CB2 3QG, England, U.K., June 1994.

[12] R. Gopalakrishna and G. Parulkar, "Efficient Quality of Service Support in Multimedia Computer Operating Systems," Tech. Rep. WUCS-94-26, Department of Computer Science, Washington University, St. Louis, Missouri, U.S.A., 3. November 1994.

[13] ITU-T Recommendation I.350, *General aspects of quality of service and network performance in digital networks, including ISDN.* Geneva, Switzerland, 6. April 1994.

[14] ITU-T Draft Recommendation Q.2931, *Edinburgh TD 155, Broadband Integrated Services Digital Network (B-ISDN), Digital Subscriber Signalling System No.2, User Network Interface Layer 3 Specification for Basic Call/Connection Control.* Geneva, Switzerland, 13. - 21. June 1994.

[15] ATM-Forum, *ATM User Network Interface Specification, Version 3.0.* Englewood Cliffs, New Jersey, U.S.A.: Prentice Hall, 1993.

[16] International Organization for Standardization, "Quality-of-Service — Basic Framework — CD Text," Tech. Rep. ISO/IEC JTC1/SC21 N9309, ISO, 9. – 13. January 1995.

[17] B. Stiller, *Flexible Protokollkonfiguration zur Unterstützung eines diensteintegrierenden Kommunikationssubsystems*, vol. 10, no. 306. Düsseldorf, Germany: VDI, 16. February 1994.

[18] T. Hutschenreuther, O. Kiese, J. Kretschmar, S. Kühn, and A. Schill, "Modell zur qualitätsgerechten Übertragung von Medienströmen," in *Anwendungsunterstützung für heterogene Rechnernetze*, (Freiberg/Sachsen, Germany), pp. 69–78, 30. – 31. March 1995.

[19] C. Vogt, R. Herrtwich, and R. Nagarajan, "HeiRAT: The Heidelberg Resource and Administration Technique, Design Philosophy and Goals," Tech. Rep. IBM-ENC 43.9213, IBM European Networking Center, Heidelberg, Germany, 1992.

[20] M. Hofmann and C. Schmidt, "Das BerKom-II-Projekt MMT," in *Klausurtagung des Institus für Telematik*, no. Interner Bericht der Universität Karlsruhe, Fakultät für Informatik, 22/95, (Oberuhldingen), pp. 105–108, 11. – 13. April 1995.

[21] A. Campbell, G. Coulson, and D. Hutchison, "A Quality-of-Service Architecture," *ACM Computer Communications Review*, vol. 24, pp. 6–27, April 1994.

[22] C. Schmidt and M. Zitterbart, "Towards Integrated QoS Management," in *First International Workshop on High Performance Protocol Architectures*, (Sophia-Antipolis, France), pp. Session 5, Paper 14, 15.-16. December 1994.

[23] K. Nahrstedt and R. Steinmetz, "Resource Management in Networked Multimedia Systems," *IEEE Computer*, vol. 29, pp. 52–63, May 1995.

[24] K. Nahrstedt and J. Smith, "An Application-driven Approach to Networked Multimedia Systems," in *18th Conference on Local Computer Networks (LCN)*, (Minneapolis, Minnesota, U.S.A.), pp. 361–367, 19. – 22. September 1993.

[25] T. Burkow, "Operating System Support for Distributed Multimedia Applications; A Survey of Current Research," Tech. Rep. Memoranda Informatica 94–57, University of Twente, Faculty of Computer Science, Twente, Netherlands, June 1994.

[26] I. Leslie, D. McAuley, and S. Mullender, "Pegasus – Operating Support for Distributed Multimedia Systems," Tech. Rep. TR 282/Pegagus-92-2, University of Cambridge, Computer Laboratory, December 1992.

[27] B. Stiller, "PROCOM: A Protocol Configuration Manager in the Function-based Communication Subsystem," in *First International Workshop on High Performance Protocol Architectures*, (Sophia-Antipolis, France), pp. Session 3, Paper 9, 15.-16. December 1994.

[28] T. Plagemann, B. Plattner, M. Vogt, and T. Walter, "A Model for Dynamic Configuration of Light-Weight Protocols," in *IEEE 3rd Workshop on Future Trends of Distributed Systems*, (Taipeh, Taiwan), pp. 100–106, 14. – 16. April 1992.

[29] D. F. Box, D. C. Schmidt, and T. Suda, "ADAPTIVE: An Object-Oriented Framework for Flexible and Adaptive Communication Protocols," in *High Performance Networking, IV*, (Amsterdam, Netherlands), pp. 367–382, IFIP Transactions C-14, North Holland, 1993.

[30] T. Roscoe, *The Structure of a Multi-Service Operating System*. University of Cambridge, Computer Laboratory, England, U.K., April 1995.

[31] W. Janzen, "Entwurf und Realisierung dynamischer Eigenschaften von FuKSS zur Verwaltung von Protokollmaschinen," in *Diplomarbeit: Universität Karlsruhe, Institut für Telematik, Germany*, November 1994.