University of Würzburg
Institute of Computer Science
Research Report Series

A Self-Organizing Concept for Distributed End-to-End Quality Monitoring

Stefan Chevul[1], Andreas Binzenhöfer[2], Matthias Schmid[3],
Kurt Tutschku[2] and Markus Fiedler[1]

Report No. 378 January 2006

[1] Blekinge Institute of Technology,
Department of Telecommunication Systems.
SE-371 79 Karlskrona, Sweden.
{stefan.chevul,markus.fiedler}@bth.se

[2] Department of Distributed Systems, Institute of Computer Science
University of Würzburg, Am Hubland, 97074 Würzburg, Germany
{binzenhoefer,tutschku}@informatik.uni-wuerzburg.de

[3] Infosim GmbH & Co. KG
Friedrich-Bergius-Ring 15, 97076 Würzburg, Germany
schmid@infosim.net

A Self-Organizing Concept for Distributed End-to-End Quality Monitoring

Stefan Chevul, Markus Fiedler
Blekinge Institute of Technology, Department of Telecommunication Systems. SE-371 79 Karlskrona, Sweden.
{stefan.chevul,markus.fiedler}@bth.se

Andreas Binzenhöfer, Kurt Tutschku
Department of Distributed Systems, Institute of Computer Science University of Würzburg, Am Hubland, 97074 Würzburg, Germany
{binzenhoefer,tutschku}@informatik.uni-wuerzburg.de

Matthias Schmid
Infosim GmbH & Co. KG Friedrich-Bergius-Ring 15, 97076 Würzburg, Germany
schmid@infosim.net

Abstract

[1] The subjective service quality as perceived by the end-user is an aspect which is often neglected in classic network monitoring. Yet it is the end-user who decides whether he will continue to use the service or not. This is especially true for real time services like Voice over IP telephony (VoIP), whose experienced quality heavily depends on the current network situation.

In this paper we present a self-organizing concept for distributed monitoring of the end-to-end quality, which will bridge the feedback gap between the user and the provider. It is based on a simple and robust measurement mechanism, which is running at the host of the end-user. As a proof-of-concept we implemented a prototype of our architecture and show the practicability of our concept using VoIP quality measurements.

1 Introduction

The operation of Quality-of-Service (QoS) providing IP networks has become highly complex. QoS monitoring, if applied, is usually carried out by rather centralized entities and only in those parts of the network where a provider is responsible for. A coordination of the monitoring among different administrative or technical domains, *e.g.* as a *monitoring service*, is rarely achieved.

In addition, the success of today's Internet results largely from the *end-to-end (E2E) concept* [1]. It empowers the end-hosts to adapt their data flow autonomously to varying load conditions. By this concept, however, the adaptation is decoupled from the network control. The network and in particular the network operator is not anymore aware of the requirements of the end-hosts, as *e.g.* their desired throughput. A direct user feedback is usually not considered [2]. As a result, a gap arises between network management,

[1]This work has been funded by the Network of Excellence EuroNGI as the specific Joint Research Project JRA.5.06 "AutoMon".

which can not address specific application, needs and a disappointed user missing QoS for his application.

Central QoS monitoring typically results in additional, complex entities at the provider. The operator has to ensure the reliability of the entities and assess their scalability. Such systems have to scale with $O(N^2)$ due to the $N \times (N-1)$ potential relationships among N end systems. In addition, relaying QoS monitoring data consumes bandwidth, delays its availability, and might be lost in case of a network failure. A decentralized QoS monitoring, located on the user's end system, would avoid these disadvantages. Furthermore, the use of a distributed, self-organizing software might reduce capital and operational expenditures (OPEX and CAPEX) of the operator since fewer entities have to be installed and operated in order to survey the service. Scalability can be achieved by re-using resident resources on an end system in conjunction with local decisions and transmission of compressed, preprocessed data.

In this paper we propose a new, distributed, self-organizing, and generic QoS monitoring architecture for IP networks. The generality of the approach consists in the facts that it can be applied for arbitrary applications, and that direct feedback to the user will be given without having to modify the monitored application. The architecture will complement today's solutions for central configuration and fault management such as HP OpenView [3] or InfoSim StableNet [4] by providing inter-working interfaces. The architecture is based on equal agents, denoted as Distributed Network Agents (DNA), which form a management overlay for the service. The self-organization of the overlay is achieved by the Kademlia algorithm [5] which builds on the concept of *Distributed Hash Tables* (DHTs). The DHT mechanism implements a distributed index for the rapid locating of other agents or resources.

As a first candidate, the architecture will use a new passive monitoring concept. The so-called *bottleneck indicator* [6, 7] obeys the end-to-end perspective of the user while being generically applicable for different applications. The suggested architecture facilitates the autonomic communication concept [8, 9] by locally determining the perceived QoS of the user from distributed measurements and by exploiting the self-organization capabilities of the DHT for structuring the overlay. The suggested self-organizing architecture is able to bridge the gap between end-user and service operator to provide quality monitored Internet access by considering the end-user's perspective and by offering a distributed monitoring service with quality feedback facilities towards network and service operators.

The paper is organized as follows. Section 2 will discuss the functional requirements for a self-organizing monitoring architecture. Section 3 describes the details of the proposed architecture, the self-organizing principle, and the measurement instrumentation. The capabilities and generality of the complete concept are demonstrated in Section 4 by investigating a case study which monitors the quality perceived by the well-known VoIP P2P application Skype [10]. Section 5 summarizes the findings and provides an outlook to further use and refinement of the monitoring concept.

2 Requirements

Our concept for distributed end-to-end quality monitoring is intended to improve the possibilities of classic network management. Therefore it needs to be practicable and offer new benefits to both the service provider and the end-user. The quality of the monitored service as perceived by the end-user will be the most important aspect of the architecture. The user should be able to apply and understand the concept without having to know what is actually happening in the network. The software will visualize the currently measured end-to-end quality in an easy-to-interprete way. We summarize the corresponding requirements for our concept in this section and use them as guidelines to deduce an efficient and reliable architecture.

First of all the concept needs to be self-organizing. This will not only reduce the configuration overhead in setting up the management architecture, but also offer a unified management environment. That is, the concept should work over multiple administrative domains using multiple network architectures without the need for a centrally organized administration. Regarding the end-user, he should not have to worry about how to set up the measurement infrastructure. Ideally, the end-user simply starts the software and automatically becomes part of the distributed monitoring system. This will also reduce the management effort involved in the monitoring of the end-to-end quality. The communication partners will be able to find each other without having to worry about things like IP-addresses, the kind of device they are using or their current physical location. The concept should also be easy to apply without the need for any kind of manual configuration by the end-user. Once the end-users are connected to each other the measurement environment will be set up autonomously and the clients will automatically start to exchange the perceived quality.

The measurement mechanism itself is one of the focal points of this paper. We will present measurements of a Skype based VoIP communication and investigate in how far the measurement concept is able to reflect the quality perceived by the end-user. The requirements to such a concept are that it should be robust against changes in the network, packet loss and delay variations without the need for synchronized clocks. It should be able to pinpoint those disturbances that actually influence the performance experienced by the end-user. Combining the observed network utility at the inlet and the outlet it will capture the damping effect of the network onto user-perceived quality from an end-to-end perspective. Instead of being limited to a specific task like the monitoring of the above mentioned VoIP connection, it should be universally applicable and adaptable to new application scenarios. In this context, it should be easy to deploy new services like video conferencing or streaming, real time chatting or content distribution mechanisms. In the following we will deduce the architecture of our self-organizing concept from the features and requirements summarized in this section.

3 Definition of the Architecture

The main goal of our self-organizing concept for distributed end-to-end quality monitoring is to be simple and versatile. The tool should be easy to install and as self-configuring

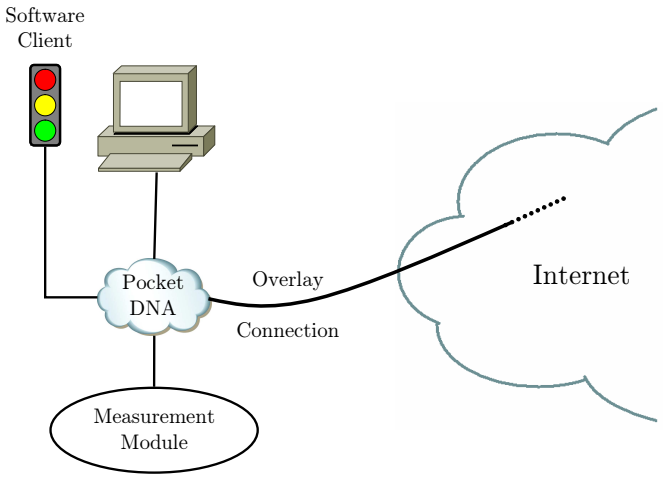

Figure 1: The main components of the architecture

as possible. The results of the end-to-end monitoring should furthermore be easy to understand by the end-user, the provider or the operator. To achieve these goals, we divided the architecture into three separate components:

- A self-organizing overlay network, which keeps the participating peers connected and enables fast and efficient searches for other users independent of their current physical location;
- A compact software client, which runs as a module on each peer and visualizes the measured performance of the network in a way that can easily be interpreted by a non experienced end-user;
- A measurement concept, which collects informations about the current state of the network and translates those values into humanly understandable terms.

The three main components of the architecture are illustrated in Fig. 1. The (mobile) end-user is connected to a self-organizing overlay network using a pocket version of our distributed network application (DNA) [11], which is explained in Subsection 3.1. The software client (*cf.* Subsection 3.2) and the measurement concept (*cf.* Subsection 3.3) are realized as modules of this Pocket-DNA. The software client captures the packets from the network and passes them to the measurement module for further analysis. The results will then be transformed into a humanly readable form by the software client.

3.1 The Self-Organizing Overlay

The main requirement of the self-organizing overlay is to enable the end-users to communicate with each other independent of their current location, their current IP address or the kind of device they are using. The idea is to keep the participating end-users connected in an autonomous way. That is, the user should be able to start the software

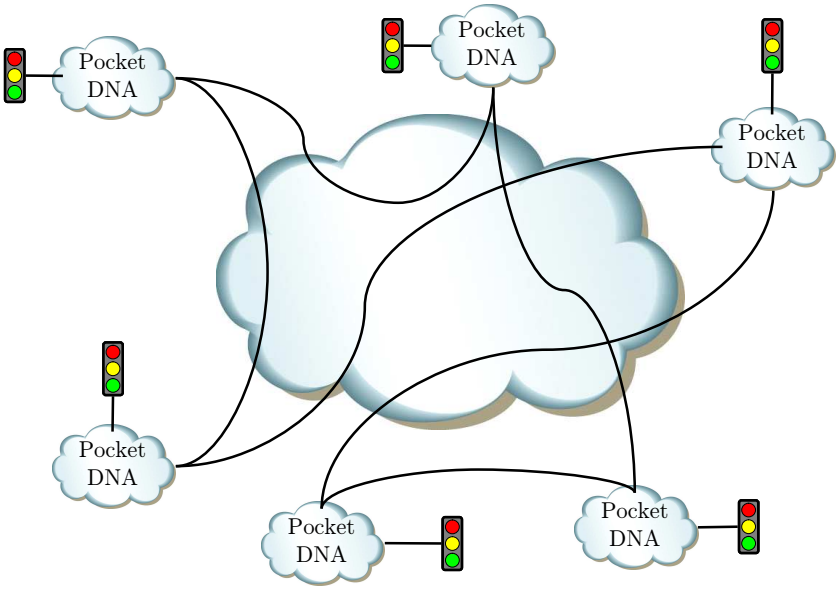

Figure 2: The Pocket-DNA overlay

on his current device and automatically become a part of the overlay network. All users in the overlay are able to search for any other user in the overlay using a simple search criterion like the nickname of the searched user. This functionality is already offered by our DNA client introduced in [11]. In this paper we use a simplified version of the DNA client, the Pocket-DNA, as a step toward a future Pocket-PC version of the management software. Fig. 2 illustrates six Pocket-DNAs building an overlay network on top of the Internet. The algorithm used to organize the overlay network is based on a modified version of the Kademlia protocol as described in [11]. Kademlia [5] is a structured peer-to-peer (P2P) network which can guarantee to find any online user within $\log(n)$ overlay hops, where n is the current size of the overlay network. Thereby it does not make any difference on what kind of device the Pocket-DNA software is running at the moment. The corresponding maintenance cost for each participating peer is to keep connections to $\log(n)$ well defined neighbors in the overlay.

3.2 The Software Client Module

The entire framework can be used to perform distributed tests and distributed monitoring for fault and performance management. In that way, the concept is able to overcome disadvantages that come along with a central management unit, like lack of scalability and reliability. It also becomes possible to monitor and provision service quality in an end-to-end manner as perceived by the user, which otherwise is rarely achieved.

The purpose of the Pocket-DNA itself is to maintain the overlay in a self-organizing way. Beyond this, it does not offer any additional functionality. The functionality itself comes with the modules which can be attached to the Pocket-DNA. The framework can,

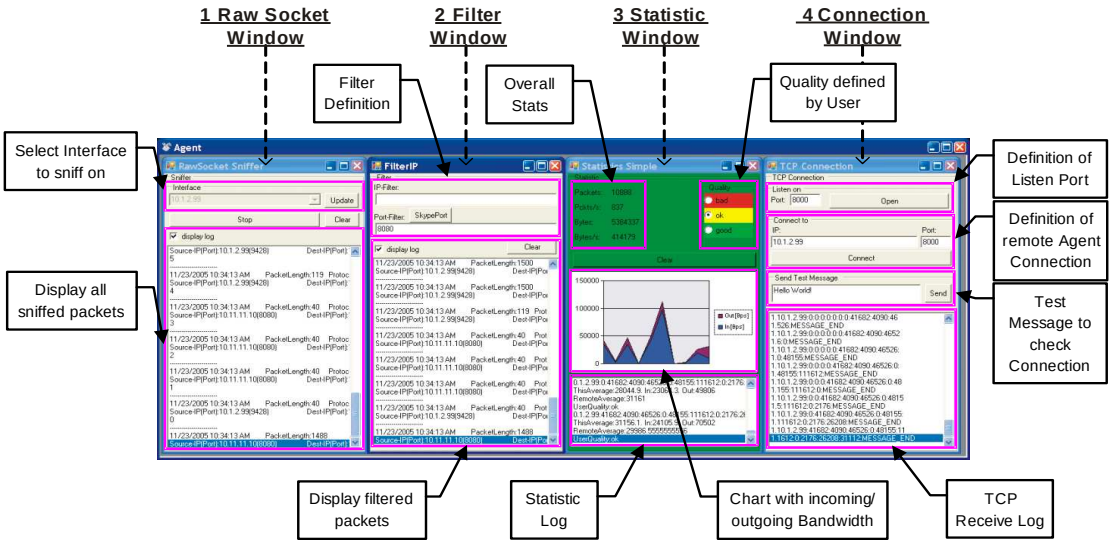

Figure 3: Screenshot of the QJudge prototype

e.g., be used to easily realize edge based services or distributed network management. In the following two subsection we describe a module which consists of a software client and a measurement concept. This module can be used to capture the end-to-end quality of a connection between two Pocket-DNA clients and to visualize the results to the user in a way which he can easily comprehend. The software client is realized as a module of the Pocket-DNA, which we call the QJudge. It represents an easy to use and extendable platform, which was written in *C#*. The main purpose of this module is to capture, filter, and analyze different statistics concerning the IP traffic of the client. The corresponding results will automatically be shared between the two communication partners. Therefore the tool has to be installed on both end hosts. After setting an appropriate traffic filter (IP-address and port) all data captured by the local QJudge module will be sent to the remote Pocket-DNA as defined in the TCP Connection-Window. The remote Pocket-DNA can then compare its locally measured statistics to the one received from the other Pocket-DNA. Information about all filtered packets, internal quality-calculations and user-quality-selections are stored in log-files for offline evaluation at a later point in time.

The two main classes used for sniffing and exchanging results are the Socket class and the TcpListener Class. These classes are included in the .NET Framework by default and can easily be accessed using *C#* as programming language in Visual Studio. After selecting which interface should be sniffed the Socket class provides all packets going over this interface byte by byte. After reading out source-/destination- IP address and port the filter decides if the packet goes into the statistic module which uses the TcpListener Class for a simple data exchange.

Fig. 3 shows a screenshot of the current version of the QJudge prototype. It consists of the following four components:

3.2.1 RawSocket Sniffer

This subwindow displays all incoming and outgoing traffic, which traverses the selected interface of the Pocket-DNA. This interface determines which connections will be monitored by the QJudge. The end-user is only able to select another interface once the QJudge has been stopped. By default the first interface card of the PC is used. Since most PCs only have one interface, this should be the desired setting for the default user.

3.2.2 Filter

In order to evaluate the performance of a specific connection, the end-user is able to apply a corresponding packet filter. In this context, the *Skype-Port* button can be used to automatically determine the port of the Skype application reading its configuration file. It is also possible to enter an IP address and/or a port manually. The filter window shows all packets which meet the given filter criteria and finally sends them to the measurement module. If both IP and port filter are empty all packets are forwarded correspondingly .

3.2.3 Statistics

The statistics window displays the results obtained by the measurement module in a way which is easy to interpret by the end-user. The part of the window above the throughput ("incoming/outgoing bandwidth") plot changes colour (*e.g.* from green = "OK" to red = "performance alarm") according to the results that will be discussed in Section 4. As this functionality builds upon the comparison of throughput statistics between source and destination, the TCP connection toward the remote communication partner is used to exchange these statistics. To be able to compare the quality as calculated by the measurement module to the quality as perceived by the end-user, it is also possible to indicate the subjective quality manually, which will be noted in the corresponding trace.

3.2.4 TCP Connection

The TCP connection is used to transmit the local measurement results to the remote Pocket-DNA. If there is no communication partner, the results are sent to the local Pocket-DNA for demonstration purposes. In order to change the communication partner on the fly, the IP address (and port number) have to be changed and confirmed with the *Connect* button. The *Send* button is only for internal test and validation purposes. The text in the input field above this button is transmitted to the remote side in order to verify the connectivity. The text below in the log view shows all data received by the remote side, while the first number in each line is the connection ID. By default port 8000 is used for incoming TCP connections.

The measurement concept used to evaluate the data captured by the QJudge is explained in the next subsection.

3.3 The Measurement Concept

In order to reflect the viewpoint of user-perceived quality from an end-to-end perspective, summary statistics of perceived throughput on a rather small time interval, ΔT, are collected by the QJudge and compared at the source and destination hosts during an observation window of ΔW. This concept called throughput histogram difference plots was first introduced in [6].

Let $\{T_p, L_p\}_{p=1}^{k}$ be a set of packets as sent by a source application or received by a destination application. From this, we obtain the values of the throughput time series as

$$R_{\mathrm{A},s} = \frac{\sum_{\forall p: T_p \in](s-1)\Delta T, s\Delta T]} L_p}{\Delta T}\,. \tag{1}$$

The impact of the network is captured by the following summary statistics:

1. *Average application-perceived throughput:*

$$\bar{R}_{\mathrm{A}} = \frac{1}{n}\sum_{s=1}^{n} R_{\mathrm{A},s}. \tag{2}$$

   A change of this parameter between source and destination ($\bar{R}_{\mathrm{A}}^{\mathrm{dst}} < \bar{R}_{\mathrm{A}}^{\mathrm{src}}$) reflects missing traffic at the end of the observation interval. That share of traffic might have been lost. The loss ratio is given by:

$$\ell = \max\left\{1 - \frac{\bar{R}_{\mathrm{A}}^{\mathrm{dst}}}{\bar{R}_{\mathrm{A}}^{\mathrm{src}}}, 0\right\}. \tag{3}$$

2. *Standard deviation of the application-perceived throughput:*

$$\sigma_{R_{\mathrm{A}}} = \sqrt{\frac{1}{n-1}\sum_{s=1}^{n}\left(R_{\mathrm{A},s} - \bar{R}_{\mathrm{A}}\right)^2}. \tag{4}$$

   A rising standard deviation ($\sigma_{R_{\mathrm{A}}}^{\mathrm{src}} > \sigma_{R_{\mathrm{A}}}^{\mathrm{dst}}$) reflects a growing burstiness of the traffic between source and destination, while a sinking standard deviation ($\sigma_{R_{\mathrm{A}}}^{\mathrm{src}} < \sigma_{R_{\mathrm{A}}}^{\mathrm{dst}}$) means a reduction of burstiness. In the latter case, the throughput histogram becomes more narrow, which means that the traffic has been shaped.

3. *Coefficient of variation:*

$$c_{R_{\mathrm{A}}} = \frac{\sigma_{R_{\mathrm{A}}}}{\bar{R}_{\mathrm{A}}}. \tag{5}$$

   This parameter represents a burstiness indicator that even takes the average throughput into account. The difference

$$\gamma = c_{R_{\mathrm{A}}}^{\mathrm{dst}} - c_{R_{\mathrm{A}}}^{\mathrm{src}}. \tag{6}$$

   denotes the absolute change of the coefficient of variation seen from the viewpoint of the receiver.

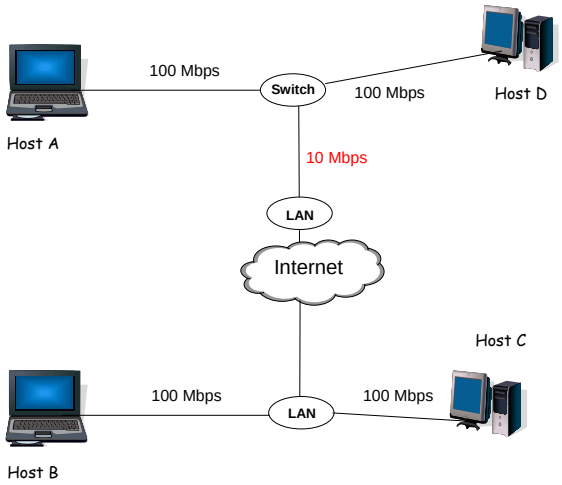

Figure 4: Measurement setup

4. *Application-perceived throughput histogram:*

$$h_{R_\mathrm{A}}(i) = \frac{\text{number of } R_{\mathrm{A},s} \in](i-1)\Delta R, i\Delta R]}{n}. \tag{7}$$

A throughput histogram difference plot with

$$\Delta h_{R_\mathrm{A}}(i) = h^{\mathrm{dst}}_{R_\mathrm{A}}(i) - h^{\mathrm{src}}_{R_\mathrm{A}}(i), \tag{8}$$

originally defined in [6], serves as a visual bottleneck indicator, illustrating throughput changes perceived by traffic on its way through the network in the following ways:

- M shape: indicates resource *sharing* with other traffic yielding an increased share of lower and higher throughputs at the receiver as compare to the sender;
- W shape: indicates traffic *shaping* towards typical throughputs;
- N shape: indicates a *saturated* bottleneck with systematically lower throughput at the receiver as compared to that at the sender.

In the next section we present a case study which monitors the quality perceived by the VoIP P2P application Skype. It effectively demonstrates the capabilities and generality of our complete concept.

4 Case Study

To show the feasibility and the simplicity of our approach, we investigate and visualize the performance of a Skype voice conference via the European research networks. Fig. 4 depicts the setup. Host A is located at Blekinge Institute of Technology, Karlskrona, Sweden and host B is located at University of Würzburg, Germany. Both hosts run

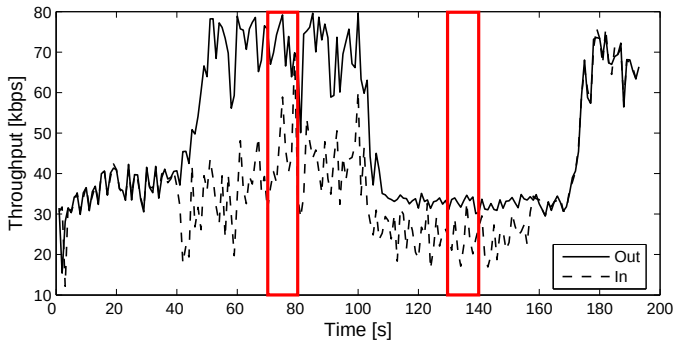

Figure 5: Throughput time plot with 9.45 Mbps background load and highlighted intervals]70 s, 80 s] and]130 s, 140 s]

Skype version 1.4.0.84 and our QJudge. Our experience is that Skype voice conferences between Karlskrona and Würzburg work very well, which is mainly the result of the corresponding research networks being over-dimensioned. To compromise this quality, host C, *cf.* Fig. 4, was used to send UDP packet streams to host D, thus turning the 10 Mbps link between the switch and the LAN into a bottleneck. In the following, we concentrate on the streams from Würzburg to Karlskrona passing the bottleneck in the same direction as the disturbing UDP stream. The observation window was chosen as $\Delta W = 10$ s and the time resolution as $\Delta T = 1$ s, while the throughput resolution ΔR was set to 3 kbps.

While tracing the traffic, a simple quality assessment of the test calls was carried out. The results were rated using a *Quality Indicator* (QI) applying a colour scheme that is well-known in the context of network and service management:

- Green (G): occurs when the user-perceived quality is good;
- Yellow (Y): occurs when there is notable disturbance, *e.g.* the perceived sound becomes robotic;
- Red (R): comes about when there is significant disturbance, *e.g.* the perceived sound quality is so bad that the conversation becomes hardly understandable;
- Violet (V): occurs when the call gets dropped due to very poor sound quality.

From each measurement, two observation intervals]70 s, 80 s] and]130 s, 140 s] were selected for statistical analysis. Table 1 shows summary statistics and corresponding QI values for the interval]70 s, 80 s] and Table 2 for interval]130 s, 140 s], respectively. From the data, we observe that the higher the load on the bottleneck, the worse the loss ratio ℓ and the quality perception become. Furthermore, there is also a tendency that the deviations of the coefficient of variation, expressed by γ, rise as the load is increased.

We now focus at two particular scenarios. The first scenario reveals the first case in which the quality was affected (QI = Y). Fig. 5 depicts a throughput time plot with

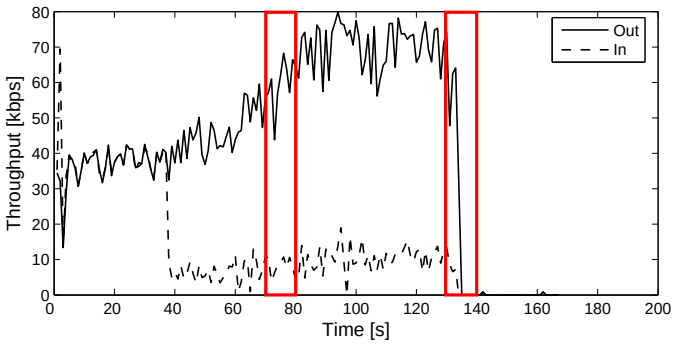

Figure 6: Throughput time plot with 9.85 Mbps background load and highlighted intervals]70 s, 80 s] and]130 s, 140 s]

Table 1: Summary statistics for interval]70 s, 80 s]

Load [Mbps]	QI	$\bar{R}_{\mathrm{A}}^{\mathrm{src}}$ [kbps]	$\bar{R}_{\mathrm{A}}^{\mathrm{dst}}$ [kbps]	$\sigma_{R_{\mathrm{A}}}^{\mathrm{src}}$ [kbps]	$\sigma_{R_{\mathrm{A}}}^{\mathrm{dst}}$ [kbps]	ℓ [%]	γ [%]
9.00	G	34.02	34.02	3.08	3.08	0.0	0.0
9.40	G	37.18	37.18	3.20	3.28	0.0	0.2
9.45	Y	70.92	45.83	4.92	10.59	35.4	16.2
9.50	R	71.89	27.52	5.39	7.32	61.7	19.1
9.55	R	68.19	22.41	10.04	4.70	67.1	6.2
9.70	R	69.25	15.50	4.23	5.66	77.6	30.4
9.80	R	56.01	7.48	6.06	2.09	86.6	17.1
9.85	V	59.64	7.95	5.97	2.56	86.7	22.2
9.90	V	71.39	8.72	5.79	3.98	87.8	37.5
9.936	V	48.53	8.01	4.91	4.53	83.5	46.4

9.45 Mbps background load and highlighted observation intervals. The first interval,]70 s, 80 s], shows that the outgoing VoIP traffic intensity from the source, Host B *cf.* Fig. 4, is larger than the incoming VoIP traffic received by the destination, Host A, thus loss is occurring. In the second observation interval the sending rate of Host B is less intense; at Host A the amount of loss is reduced. This observation is explained by Skype's own behaviour; it was noticed that Skype changed the audio codec during the measurement. This behaviour not only affected the traffic pattern but also the end-user-perceived QoS; the sound became robotic, although conversation was still possible. The second scenario treats the first case in which a call is dropped (QI = V). This behavior is visible in interval]130 s, 140 s] in Fig. 5. From Table 2, we observe a very large amount of loss (more than 80 %) and considerable changes in the coefficient of variation. Furthermore, we note that for 9.90 Mbps background load, the call is already broken before the interval]130 s, 140 s] and therefore no data is provided.

Table 2: Summary statistics for interval]130 s, 140 s]

Load [Mbps]	QI	$\bar{R}_{\mathrm{A}}^{\mathrm{src}}$ [kbps]	$\bar{R}_{\mathrm{A}}^{\mathrm{dst}}$ [kbps]	$\sigma_{R_{\mathrm{A}}}^{\mathrm{src}}$ [kbps]	$\sigma_{R_{\mathrm{A}}}^{\mathrm{dst}}$ [kbps]	ℓ [%]	γ [%]
9.00	G	32.46	32.46	1.76	1.95	0.0	0.6
9.40	G	36.27	36.36	3.28	3.85	0.2	1.5
9.45	Y	32.80	23.92	1.73	4.27	27.1	12.6
9.50	R	39.79	18.76	4.13	3.48	52.9	8.2
9.55	R	66.82	31.72	4.92	5.78	52.5	10.9
9.70	R	69.48	20.98	3.91	7.78	69.8	31.5
9.80	R	70.31	12.92	7.14	6.07	81.6	36.8
9.85	V	22.69	2.23	29.54	3.63	90.2	32.6
9.90	V	0	0	0	0	N/A	N/A
9.936	V	69.48	11.02	7.66	3.42	84.1	20.0

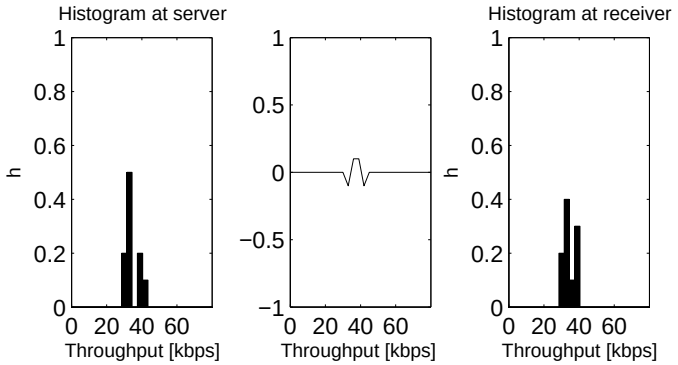

Figure 7: Throughput histograms and difference plot with 9.40 Mbps background load, QI = G

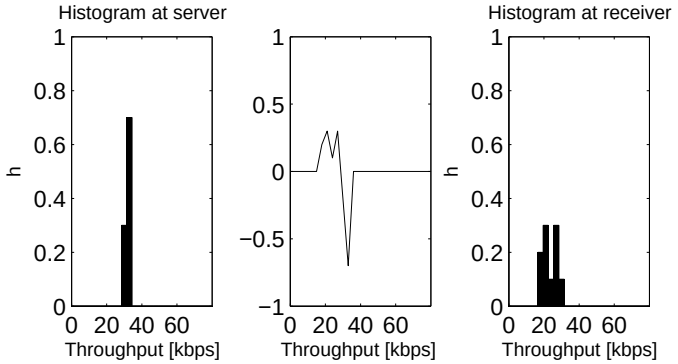

Figure 8: Throughput histograms and difference plot with 9.45 Mbps background load, QI = Y

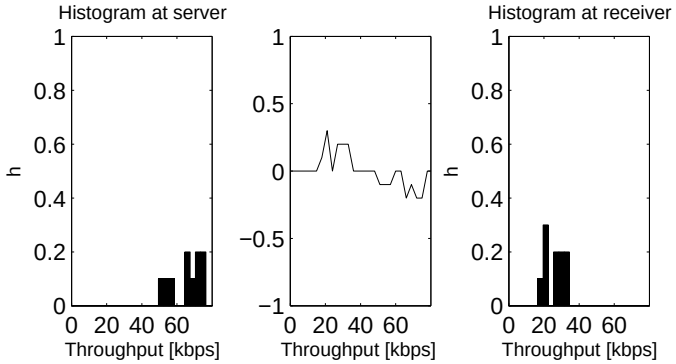

Figure 9: Throughput histograms and difference plot with 9.50 Mbps background load, QI = R

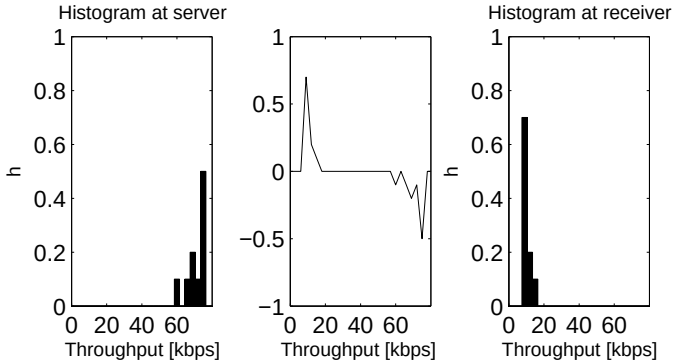

Figure 10: Throughput histograms and difference plot with 9.85 Mbps background load, QI = V

In order to visualise the connection between statistics and quality perception, we depict the throughput histogram difference plots for all different QI values in Fig. 7 to Fig. 10. We note that the bottleneck indicator (middle in Fig. 7 to Fig. 10) changes its shape with rising background traffic: the two peaks making the N form move away from each other, indicating a growing mismatch between offered traffic and carried traffic. The latter becomes less as the background traffic rises, which is seen from the right diagrams. From the left diagrams, we also notice that Skype increases the sending rate. Thus, the obvious discrepancy between offered traffic and carried traffic affects the user-perceived quality.

5 Conclusion

In this paper we proposed a simple and versatile self-organizing concept for distributed end-to-end quality monitoring. Both architecture of this concept and a prototype were presented. The concept was further demonstrated by a case study in which the user-perceived quality of the well-known VoIP P2P application Skype was monitored. The influence of network overload on relevant summary statistics was illustrated and discussed. Throughput-related statistics have been used successfully to visualize critical network impacts on user-perceived quality. It has been identified that the throughput variations increased and high amount of loss occurred as the network overload intensified. For visualization of the user-perceived quality rating, simple colour coding, as known from Network Management Systems has been applied.

Future work will investigate the impact of traffic variations on the user-perceived quality and the quantitative assessment of quality problems using the Network Utility Function [7]. It will also consider the adaptation of networks based on the information provided by QJudge, *e.g.* by pinpointing critical subnetworks.

References

[1] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems*, vol. 2, pp. 277–288, November 1984.

[2] M. Fiedler (ed.), "EuroNGI Deliverable D.WP.JRA.6.1.1 – State-of-the-art with regards to user-perceived Quality of Service and quality feedback," May 2004. URL: http://www.eurongi.org/, http://www.ats.tek.bth.se/eurongi/dwpjra611.pdf.

[3] HP OpenView Management Software. URL: http://www.openview.hp.com/.

[4] InfoSim GmbH & Co. KG. URL: http://www.infosim.net/.

[5] P. Maymounkov and D. Mazires, "Kademlia: A peer-to-peer information system based on the xor metric," in *Revised Papers from the First International Workshop on Peer-to-PeerSystems*, pp. 53–65, Springer-Verlag, 2002.

[6] M. Fiedler, K. Tutschku, P. Carlsson, and A. Nilsson, "Identification of performance degradation in IP networks using throughput statistics," in *Providing Quality of Service in Heterogeneous Environments. Proceedings of the 18th International Teletraffic Congress (ITC-18)* (J. Charzinski, R. Lehnert, and P. Tran Gia, eds.), (Berlin, Germany), pp. 399–407, September 2003.

[7] M. Fiedler, S. Chevul, O. Radtke, K. Tutschku, and A. Binzenhöfer, "The Network Utility Function: A practicable concept for assessing network impact on distributed systems," in *19th International Teletraffic Congress (ITC19)*, (Beijing, China), pp. 1465–1474, September 2005.

[8] Autonomic Communication (IST FP6 Project). URL: http://www.autonomic-communication.org/.

[9] IBM Autonomic Computing. URL: http://www.research.ibm.com/autonomic/.

[10] S. A. Baset and H. Schulzrinne, "An analysis of the skype peer-to-peer internet telephony protocol," Tech. Rep. CUCS-039-04, Computer Science Department, Columbia University, New York, NY, 9 2004.

[11] A. Binzenhöfer, K. Tutschku, and B. auf dem Graben, "DNA – A P2P-based Framework for Distributed Network Management," in *Peer-to-Peer-Systeme und -Anwendungen*, (GI/ITG Work-In-Progress Workshop in Cooperation with KiVS 2005, Kaiserslautern), March 2005.