University of Würzburg
Institute of Computer Science
Research Report Series

A Novel Approach to Verify End-to-End Connectivity in an Autonomic Way

Andreas Binzenhöfer, Daniel Schlosser, and Björn auf dem Graben

Report No. 383 April 2006

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

A Novel Approach to Verify End-to-End Connectivity in an Autonomic Way

Andreas Binzenhöfer, Daniel Schlosser, and Björn auf dem Graben
Department of Distributed Systems, Institute of Computer Science
University of Würzburg, Am Hubland, 97074 Würzburg, Germany
{binzenhoefer,schlosser,adgraben}@informatik.uni-wuerzburg.de

Abstract

Computer networks are increasingly becoming more complex. As a consequence unacceptable user-perceived quality is often blamed on the network. However, it requires highly trained professionals to verify the connectivity between two communication partners. The concept of autonomic networks has been proposed in an effort to make such networks easier to manage. In this context, we present a plug-and-play mechanism to check the connectivity between two end points of the network. It will significantly decrease the amount of mundane work, which is usually involved in such a task. The main goal is to autonomically measure the current conditions between two communication partners and to visualize it in an easy to interpret way. The results can be used to indicate whether the current state of the network supports specific services like VoIP calls, videoconferencing, or file transfers.

1 Introduction

The complexity of computer networks has constantly grown during the last decades. The protocols and algorithms, which we use today, are still based on assumptions, which were accurate in the 70s and 80s. The challenges, the threats, and the vulnerabilities of those systems have gone beyond what current architectures can deal with. The fragile nature of how the deployed entities are put together leads to numerous weak points. As a consequence we require new mechanisms, which are more reactive to failures and errors than it is now with the human incorporated. That is, we need autonomic systems, which are more self-managing, self-configuring and self-diagnosing. The degree to which a system can be described as being autonomic is the inverse of the degree of which a human being participates in the control loop.

One of the major problems of todays distributed applications is to find the root cause in case of a failure or decreased functionality. This is especially difficult since there are multiple areas of responsibility involved. Usually the blame is shifted from application level via other levels like the operating system all the way down to the network itself. On account of this a considerable part of the work of network administrators consists in proving that it is not the fault of the network. As a first step to decrease the involvement of the human in the control loop we therefore present a novel approach to verify end-to-end connectivity in a more autonomic way. It maps the current state of the network to a result which can easily be understood without the need of a well trained network professional. The results can also be used to indicate whether the current end-to-end quality supports a specific service like a VoIP call, a videoconference, or ftp file transfers. The architecture consists of different modules which have been integrated into our autonomic distributed network management prototype [1].

The remainder of this work is organized as follows. Section 2 summarizes the overall architecture and how the modules can be integrated into it. It also gives a detailed description of each module. Section 3 specifies our autonomic measurement concept and illustrates how to map the results to a traffic light scheme. The findings and experiences obtained from our prototype (c.f. Section 4) are presented in Section 5. Section 6 finally concludes the paper.

2 Description of the Architecture

The main architecture consists of two parts. An overlay network which provides the plug-and-play framework for distributed network management [1] and the modules which offer the measurement functionality. The overlay network is used to setup the measurement architecture in a self-configuring way. The modules are used to perform the actual measurements and to visualize the results in an easy to understand way.

2.1 Network Management Overlay

In [1] we presented a distributed network management overlay. It consists of our distributed network application (DNA) which runs on several end-hosts as shown in Figure 1. It is used to support the central network manager by extending its view on the network, performing distributed tests and offering scalable services which do not suffer from a single point of failure. The main requirement of the self-organizing overlay is to enable the peers 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 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.

Fig. 1 illustrates six DNAs building an overlay network on top of a company network. The algorithm used to organize the overlay network is based on a modified version of the Kademlia protocol as described in [1]. Kademlia [2] 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. The corresponding maintenance cost for each participating peer is to keep connections to $\log(n)$ well defined neighbors in the overlay.

In our case we want to perform end-to-end measurements to verify the connectivity between two specific hosts, say host A and host B. All the administrator has to do is to start the DNA prototype on both end hosts. This can either be done remotely, e.g. using the Terminal Services in Windows Server 2003, or on the spot launching the application from a USB stick. Host A and host B will both automatically become a

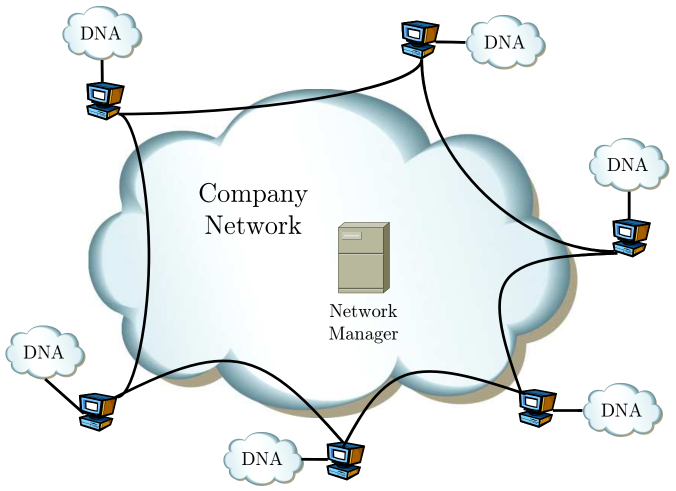


Figure 1: The distributed network management overlay

part of the overlay without any configuration overhead. A connection between the two hosts can then be established via the search functionality of the overlay network without having to know the IP-address or the exact location of the communication partner. The measurements themselves will be done using the modules attached to the DNAs as described in the following subsection.

2.2 Verification of the End-to-end Connectivity

The verification process of the end-to-end connectivity is divided into two subparts. The first step is to verify that the communication between two specific end-points of the network is possible at all. The second task is to examine whether the current quality of the network connection supports a specific service. This can e.g. be used to check whether the current quality suffices to make a VoIP call, a videoconference or a file transfer. The implementation is realized using three different modules. which are integrated into our DNA client as shown in Fig. 2. Since a significant amount

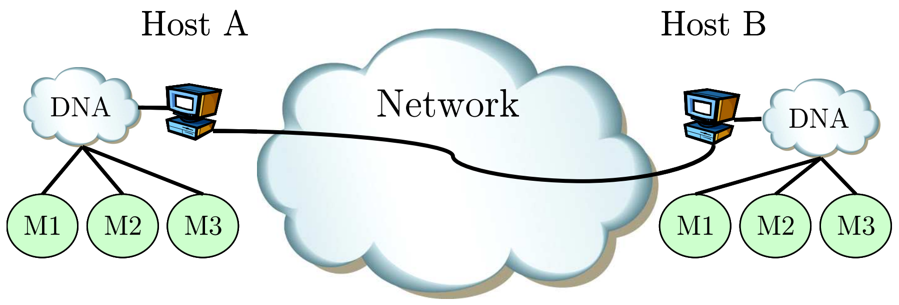


Figure 2: The modules providing the measurement functionality

of assumed network errors does indeed have a local cause, module one checks the local configuration and determines whether the device is ready to use the network. The second module measures the characteristics of the traffic sent into the network and exchanges this information with the corresponding module of the communication partner. This way Host A is able to correlate its received traffic with the one sent by host B and the other way around. The results are mapped to a simple traffic light scheme, which is easy to understand by any end-user. The third module represents a traffic generator, which is able to emulate specific services like Skype calls or traffic with a constant bitrate. The three modules will be described in detail in the following subsections. How to map the measurement results to a traffic light scheme is shown in Section 3.

2.2.1 M1: Ready for Use

The first step in verifying the end-to-end connectivity is to exclude errors which actually have a local cause. The idea is to automate all the things an administrator usually does when he checks the local configuration of an end host. This might sound trivial, but has an amazing effect on the satisfaction of the customer with the network. During field tests with our prototype end-users did put less blame on the network once they noticed that the lack of connectivity was actually caused by a local misconfiguration. In terms of autonomic networking it also greatly reduces the amount of mundane work required by a human network specialist. In fact, there are a lot of routine tests which can be automated. The following list gives some exemplary tests, which can easily be automated. It is by far not exhaustive.

- *NICStatus:* Collect information about the network interface card (NIC) and its current state. This is mainly used to eliminate the possibility of hardware failures or problems with the driver.
- *NetConnectionStatus:* Check the current connection status of the NIC. Causes for an error include a cable that is not plugged in, or a network interface card still running an authentication process.
- *PingLocalHost:* The functionality of the TCP/IP stack is validated by sending a ping request to the loopback address (127.0.0.1).
- *IPConfiguration:* Verify that a valid IP address is assigned to the NIC and check if the associated gateway is on the same subnet as the IP address.
- *DNSConfiguration:* See that at least one DNS server is assigned and can be reached by a ping request. The functionality of the DNS server is tested performing a predefined DNS lookup and optionally a reverse lookup.
- *PingOwnIP:* To exclude a communication problem between the operating system and the NIC, the IP address assigned to the adapter is pinged.
- *PingWellKnownHosts:* Finally, send a ping request to a list of predefined well known hosts. If a specified number of hosts in this list does not respond in time, the test returns an error.

In addition to those general tests, one could also include some more operating system specific test. Since our prototype is running on Windows, we included the following tests into our prototype.

- *DHCPLease:* In case of using DHCP, check the IP address to ensure that is not using an Automatic Private IP Address (APIPA). This might occur, if the server has no more capacities to provide a new IP address or if the user participates in an encrypted wireless LAN (e.g. WEP) using an invalid key.
- *EventViewer:* Browse the "Event Viewer Log" for error events caused by TCP/IP or DHCP.
- *HostsAndLmHosts:* Search for syntax errors in both the HOSTS and the LMHOSTS file. If one of these files contains a wrong entry the resolution of the corresponding name fails.
- *RoutingTable:* The Windows routing table is divided into a dynamic and a persistent part. This test pings the gateways of both tables and reports an error message, if one of them is not reachable.

All above tests can easily be automated and send a corresponding error message to the end-user in case of a failure. Once the local connectivity is ensured on both sides, the quality of the end-to-end connectivity can be examined.

2.2.2 M2: Traffic Light

The traffic light module is the heart of our architecture. It measures the network traffic on two specific end-host, exchanges information about the measurements and analyzes the effects of the network on the user perceived quality. The results will be visualized in a traffic light scheme. That is, both end-user will see two traffic lights showing the current quality of their uplink and downlink respectively. The quality can either be good (green), acceptable (yellow) or bad (red). In this section we concentrate on the measurement methodology and the exchange of the results. Section 3 explains the analysis of the results and the mapping to the traffic light scheme for different kind of services.

There are only two things which need to be configured before the module can start with the measurements: The overlay name of the remote host and R_T, the time between the exchange of two results (default 10s). The interval R_T indirectly determines how frequently the color of the traffic light is updated. Therefore, we also include the optional possibility to exchange results every R_P packets. The two options can be combined to send results every R_P packets but at least every R_T seconds. If a specific service should be monitored, the port of the service also has to be specified. However, this can be done automatically for predefined services.

Using the search functionality of the overlay network, the traffic light module is able to automatically determine the IP-address of the communication partner by its overlay nickname. While the module is running, it captures all packets which match this IP-address and the optional port. For each of these packets it records the following three values: The IP-ID, the timestamp and the size of the packet. Based on all outgoing packets captured within a measurement interval (as specified by R_T and R_P) host B creates a result summary as shown in Table 1 and sends it back to host A.

Table 1: Structure of the summary packet

Name	Size	Description
SP_S	1 bit	0 for a summary packet, 1 for a summary packet response
C	2 bit	Color of downlink traffic light of host B
encoding	1 bit	0 if $\Delta t^B_{out}(i)$ are 16 bit integers, 1 if $\Delta t^B_{out}(i)$ are 32 bit integers
S_{ID}	4 bit	summary packet ID
$t^B_{out}(1)$	8 byte	time stamp of first outgoing packet
$\Delta t^B_{out}(i)$	2/4 byte	times between outgoing packets i and $i-1$, if $i > 1$, 0 otherwise
$ID^B_{out}(i)$	2 byte	IP-ID of packet i

The first bit SP_S determines whether the packet is a summary packet or an acknowledgment. The color of the uplink traffic light of host A can immediately be set to the color C specified by host B in the summary packet. The remaining information in the packet suffices to recreate the timestamps $t^B_{out}(i)$ and the IDs $ID^B_{out}(i)$ of all outgoing packets of host B. Host A will then compare these values to the corresponding timestamps $t^A_{in}(i)$ and $ID^A_{in}(i)$ to derive the color of its dowlink traffic light. A detailed description of the corresponding algorithms is given in Section 3. To protect the exchange of the summary packet against packet loss, host A sends an acknowledgment message to host B. This message includes the median of the measured one way delays of link AB m_{AB}, the jitter, and the throughput.

2.2.3 M3: Traffic Generator

The traffic light module is a very useful tool to check, whether an already existing application achieves a predefined perceived quality of service or not. However, in many situations it is interesting to know, whether an existing network is suitable for a new application or not. In this case a passive measurement approach would be very unsatisfying since it would be necessary to actually install the application before the measurements can be done. In order to overcome this problem a traffic generator is added to the traffic light module. This enables the user to estimate the perceived quality of a network for future applications.

The only thing needed to be known is the structure of the traffic, i.e. the distribution of the inter arrival time of the packets and the distribution of the payload. The traffic generator of the traffic light module is initialized with those two distributions. From this input two independent random number streams are generated. The first one is used to define the time between two packets. The second one determines their payload. Thus, we are able to emulate every traffic pattern, which can be described by two independent distributions, e.g. ftp or VoIP. Using the traffic light scheme it is then easy to decide, whether the user perceived quality is acceptable or not. The possibility to use fixed predefined random streams or replay captured traffic streams is scheduled for future work.

3 Analyzing and Visualizing the Measurements

The traffic light module measures and exchanges the timestamps and IDs of all packets sent during a specific interval. In this section we will show how this information can be used to calculate one way packet loss, jitter and delay and how to map those values to a traffic light scheme for different services. Without loss of generality we will concentrate on host A. It is able to capture the IDs $ID_{in}^{A}(i)$ and the timestamps $t_{in}^{A}(i)$ of all incoming packets according to its own clock. From the summary packet sent by host B it also knows the IDs $ID_{out}^{B}(i)$ and the timestamps $t_{out}^{B}(i)$ according to the clock of host B. Note that host A will calculate the different measures for its downlink (link BA) while the quality of the uplink (link AB) is determined by host B.

3.1 One Way Packet Loss

The calculation of the packet loss is straight forward. Host A simply uses the packet IDs to determine which of the packets sent by host B never arrived at host A. That is, it regards all incoming packet IDs $ID_{in}^{A}(j)$ with

$$ID_{out}^{B}(1) \leq ID_{in}^{A}(j) \leq ID_{out}^{B}(i_{max})$$

and calculates the percentage of missing IDs. Note that it is also possible to pinpoint exactly which packets have been lost. This can e.g. be used to find out whether packet loss has occurred in bursts or more randomly. In the following we will assume that $i = 1, ..., i_{max}$ is the index of the ith successfully transmitted packet from host B to host A.

3.2 One Way Jitter

In order to calculate an estimate for the one way jitter over time, we draw on the formula applied in the real-time transport protocol (RTP) [3]. From the time stamps in the summary packet we calculate the time between outgoing packet i and $i+1$ at host B as:

$$\Delta t_{out}^{B}(i) = t_{out}^{B}(i+1) - t_{out}^{B}(i).$$

Accordingly, the inter arrival time of packet i and $i+1$ at host A is:

$$\Delta t_{in}^{A}(i) = t_{in}^{A}(i+1) - t_{in}^{A}(i).$$

We initialize the jitter on link BA with $j_{BA}(1) = 0$ and estimate the current jitter as

$$j_{BA}(i) = j_{BA}(i-1) + \frac{\left|\Delta t^B_{out}(i) - \Delta t^A_{in}(i)\right| - j_{BA}(i-1)}{16} \qquad \text{for } i = 2, ..., i_{max} - 1.$$

The gain parameter is set to $\frac{1}{16}$ since this gives a good noise reduction ratio while maintaining a reasonable rate of convergence [3].

3.3 One Way Delay

The one way delay is obviously the value, which is most difficult to measure. The problem is that an accurate time synchronization is absolutely essential for the measurement of the one-way delay. There are different ways to achieve synchronized clocks [4]. The most prominent examples are atomic clocks, GPS and the network time protocol (NTP) [5]. Atomic clocks yield precisely synchronized times, but need to be installed at each end host at a high cost. Using the Pulse Per Second (PPS) signal from GPS receivers is also expensive and additionally requires a clear view of the sky. Finally, NTP allows all clocks to be synchronized over a local network or even over the Internet. The precision of NTP depends on the quality of the network, i.e. on things such as the network delays. The typical error for local networks is about 1ms and up to 10ms for the Internet. A good summary of NTP can be found in [6]

Let Θ be the difference in milliseconds between the clocks of host A and host B. Without loss of generality we assume that $Clock_A = Clock_B - \Theta$. If $D_{BA}(i)$ is the measured delay of the ith successfully transmitted packet from host B to host A, then the real delay would be $D_{BA}(i) + \Theta$. For unsynchronized clocks, we have to assume that Θ might be quite large, which would render a one way delay measurement impossible. In this case, one usually has to measure the round trip delay, divide it by two and use the outcome as an estimate for the one way delay.

In general, we do not know whether the clocks of host A and host B are synchronized or not. Therefore host A initially calculates the one way delay of the ith packet on link BA as

$$D_{BA}(i) = t^A_{in}(i) - t^B_{out}(i).$$

Host B does the same for the reverse direction

$$D_{AB}(i) = t^B_{in}(i) - t^A_{out}(i).$$

As long as all delays measured this way are positive, Θ has to be smaller than the shortest actual one way delay. In this case we assume synchronized clocks and take $D_{BA}(i)$ as a direct estimate for the one way delay on the downlink of host A. In the worst case this estimate is twice as large as the actual delay. However, given the precision of synchronized clocks, this scenario is very unlikely. Furthermore, we do not need a perfectly accurate measurement in order to map it to a traffic light scheme.

If otherwise at least one of the measured delays is negative, we assume unsynchronized clocks and $\Theta >> D_{BA}(i)$. In this case we cannot rely on our measurement. However, instead of performing additional active round trip time measurements, we can easily adjust our passive results. Note that only one of the two communication partners will observe negative delays. For this reason each host calculates the median m_{BA} (or m_{AB} respectively) of the measured one way delays on its side and inserts it into the acknowledgment of the summary packet. If one of these values is negative, it will recalculate its estimates of the one way delays. In case $m_{BA} < 0$ this results in

$$D'_{BA}(i) = D_{BA}(i) + \frac{m_{AB} - m_{BA}}{2}.$$

The equation assumes symmetric one way delays and readjusts the measured values in such a way, that the median one way delay is equal in the uplink and the downlink. Note that this maintains the difference between the individual delays.

Another problem, which has not yet been discussed is that computer clocks tend to tick at different rates [7]. It is not untypical for two unsynchronized clocks to drift apart by as much as 60 milliseconds within 10 minutes. To estimate and remove the relative drift between two clocks we rely on the algorithms presented in [8]. Looking at a set of one way delays, we first need to determine whether the clock drift shows a positive or a negative trend. This can be done using a robust linear fit to the measured one way delays. If the slope of the fit is positive, we assume a positive trend. If the slope is negative, we assume a negative trend respectively. We then apply the hypothesis test described in [8] for the corresponding slope in order to determine whether we are actually dealing with a clock drift or not. If so we determine the size of the trend by applying a robust linear fit to the de-noised one way delays. That is, we first de-noise the one way delays $D_{BA}(i)$ by dividing them into $\lfloor\sqrt{i_{max}}\rfloor$ intervals and taking the minimum delay of each interval. Then we estimate the $clock_{drift}$ as the median of all the pairwise slopes between the different minima. Finally we remove the drift by adjusting the one way delays as follows:

$$D'_{BA}(i) = D_{BA}(i) - clock_{drift} \cdot \left(t^A_{in}(i) - t^A_{in}(1)\right).$$

A more detailed description can be found in [8].

3.4 Calculating the Traffic Light

The process of mapping the measured network characteristics to a traffic light color indicating the current quality of a specific service is not a trivial one. The reason is that different services react differently to changes in the network. While real time services have strict delay and packet loss requirements, the quality of a file transfer mainly depends on the average throughput. Utility functions [9] are a good way to translate network dynamics into humanly understandable terms. They take parameters like packet loss, jitter and delay as input and deliver a number between zero and one, which reflects the current user perceived quality. The problem is that those functions have to be adapted for each service.

The same can be achieved with a simple function which is defined in sections or a multidimensional matrix, which maps the measured values to a specific color. In the

	delay[ms]	packet loss[%]	jitter[ms]
green	<75	<1	<10
yellow	<200	<3	<25
red	>200	>3	>25

Table 2: Traffic Light Colors for VoIP

current version of our prototype we decided to use simple tables to determine the color of the traffic light. The prototype offers the possibility to define new services and to specify which values correspond to which color. For VoIP connections, e.g., we defined Table 2 according to the ITU-T recommendation G.114 [10]. There are numerous algorithms and models for real time services which can also directly be integrated into our prototype. The E-model [11], e. g., is a well established computational model that uses measured network parameters to predict the subjective quality of voice calls. The output of the model is the transmission rating factor (R-factor), which lies between zero (poor) and 100 (excellent) and can easily be mapped to our three color scheme. [12] gives a good overview of the E-model. Finally, in [13] a set of measurements that can be used to derive a media delivery index is described in detail. It indicates the instantaneous and longer term behavior of networks carrying streaming media such as MPEG video.

It is significantly easier to adapt the traffic light scheme to services, which do not have real time requirements. Those services usually have objective criteria by which they can be evaluated. The quality of file transfers, e.g., can be derived from one single parameter, the achieved throughput.

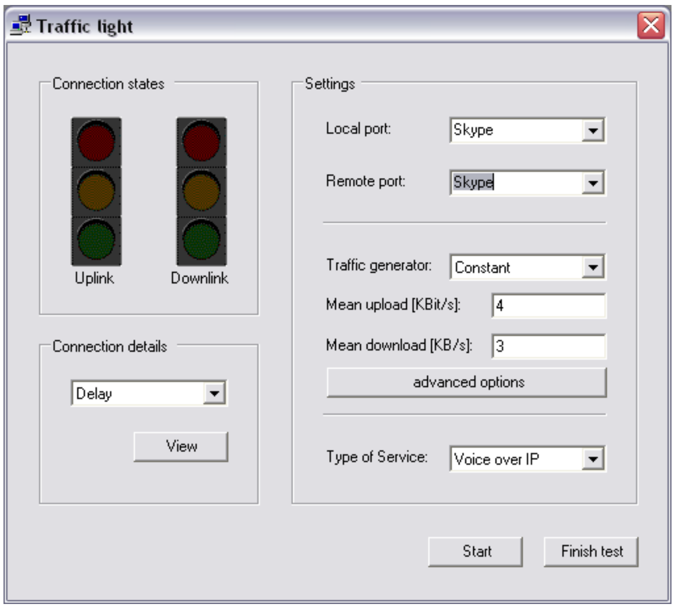
4 Implementation of the Prototype

In [1] we presented the DNA framework and a corresponding prototype which is implemented in the .NET framework [14]. The modular concept of the prototype allows an easy integration of new functionality. The traffic light module described in this paper was realized as an extension to the DNA framework.

The DNA framework itself was designed to be as system independent as possible. The functionality of the traffic light module, e.g., is especially useful on mobile devices. If the user manually changes his connection type or experiences fluctuations of his radio reception, the traffic light immediately signals whether the current quality of the network is still acceptable for a given service. To offer this functionality on heterogeneous devices we did not use any system dependent network sniffing library in the module. Instead we used the .NET socket class with a raw socket scheme which allows us to sniff packets at the IP-layer. This means that the timestamps generated with this approach differ slightly from the ones generated with a kernel level sniffer like those depending on the pcap library. However, a measurement at the IP-layer complies better with our intention to capture user perceived quality. The reason is that the time at which the kernel notices the incoming data is not necessarily the same time the data is handed over to the application. This time span may differ depending on the used operation system.

The calculated user perceived quality for both directions is displayed as a traffic light scheme, as shown in Figure 3. The figure also shows the options available to configure the traffic light module and the traffic generator. The first thing that needs to be set up before the evaluation can be started is which data flow should be monitored. This is done by specifying the IP address of the destination host and optionally the port number used by the application. The IP address of the communication partner is automatically retrieved by the DNA framework. Thus it is only necessary to define which ports the application uses on the two participating hosts in case a specific service should be monitored. Both ports can be specified via a drop down combo box, which already lists some well known or accessible port numbers, e.g. the port number for ftp and Skype. If the two end hosts are not already communicating, the module also offers the possibility to actively generate traffic. The user is able to choose from a number of predefined traffic patterns or specify its own pattern. In order to keep the user interface simple, the detailed configuration of the traffic generator is separated from the main window, shown in Figure 3.

Finally a more technical oriented user has the possibility to obtain detailed charts for the measured network parameters like one way delay, jitter, throughput, and packet loss.



Local port: Skype
Remote port: Skype
Traffic generator: Constant
Mean upload [KBit/s]: 4
Mean download [KB/s]: 3
Type of Service: Voice over IP
Delay

Figure 3: Snapshot of the main user interface of the traffic light module

5 Prototype Study and Results

To validate the feasibility and the practicability of our approach we set up a small testbed environment in our laboratory. As shown in Figure 4 it consist of two end host A and B running Windows XP and a Linux server routing the traffic between the two end hosts. Our prototype was installed on host A and host B, constantly monitoring the traffic exchanged between the two. Furthermore, we installed NISTNet [15] on the

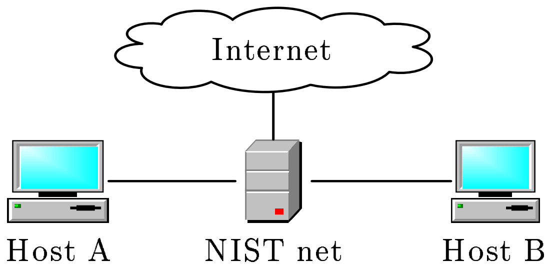


Figure 4: Setup of the testbed

Linux machine to be able to emulate typical performance dynamics of IP networks like one way delays, packet loss, jitter or asymmetric bandwidths. If not stated otherwise, we used 100Mbit/s links and a 30ms one way delay in both directions between host A and host B.

To study the influence of unsynchronized clocks, we first have a closer look at the actually measured one way delay between host A and host B. To do so, we monitored a Skype VoIP call since it periodically sends packets every 30ms. NISTNet was configured to deterministically delay the packets between A and B by 30ms. Figure 5 shows that the actual results for the measured one way delays differ from the expected values. At first, we notice that all delays are negative. The reason is that the clock of host A was running more than 18 seconds behind that of host B at the time of the measurement. In

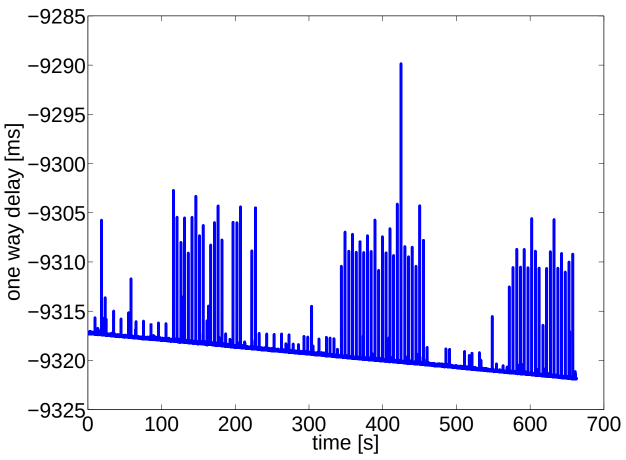


Figure 5: Actually measured delays

this case it is also obvious that the clock of host B is running slightly faster than that of host A. Regarding the figure, the clocks seem to drift apart by about 5ms during the 11 minute measurement. Our prototype recognized the negative delays as well as the skewness of the clock and adjusted both according to Section 3.3.

The adjusted one way delays are visualized in Figure 6. Now, all one way delays are

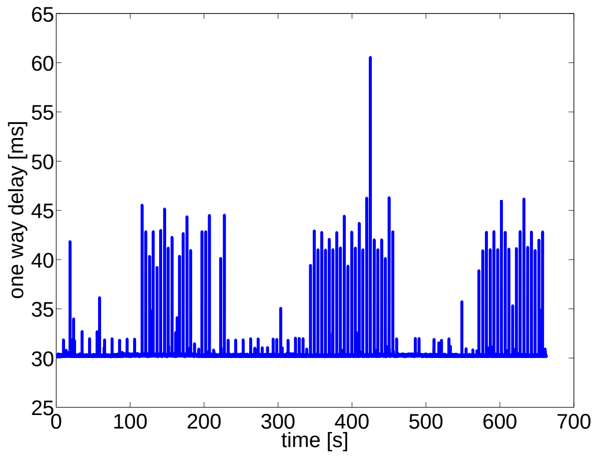


Figure 6: Adjusted one way delays

positive and the minimum delay is slightly above 30ms which corresponds to the actual physical delay between host A and host B plus the additional 30ms introduced by the NISTNet router. Note that this scenario nicely demonstrates how our prototype adjusts the measured one way delays. In other scenarios the drift of the clocks cannot always be identified that easy. See [8] for a more comprehensive discussion.

To verify the ability of our measurement environment to detect jitter we introduced Gaussian (normal) distributed jitter with a varying standard deviation. In particular,

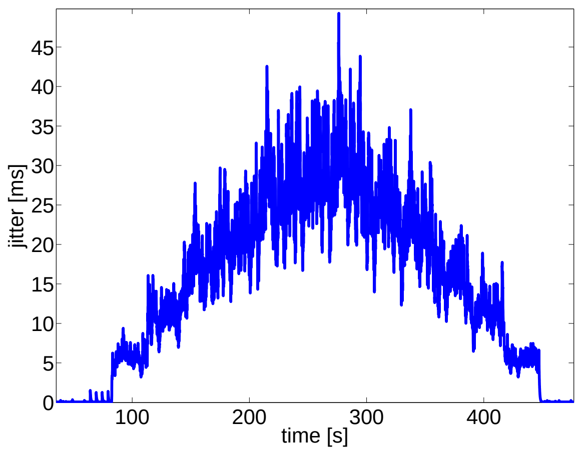


Figure 7: Measured jitter

we used a mean delay of 30ms and increased the standard deviation by 5ms every 30 seconds until we reached a standard deviation of 30ms. Then we decreased the standard deviation again by 5ms every 30 seconds until we reached a standard deviation of 0ms. Figure 7 plots the measured jitter values over time. The algorithm is able to capture the increase and the decrease of the jitter, while the absolute values oscillate more for higher jitter values.

Finally, we have a closer look at the measured packet loss. Due to the fact that host

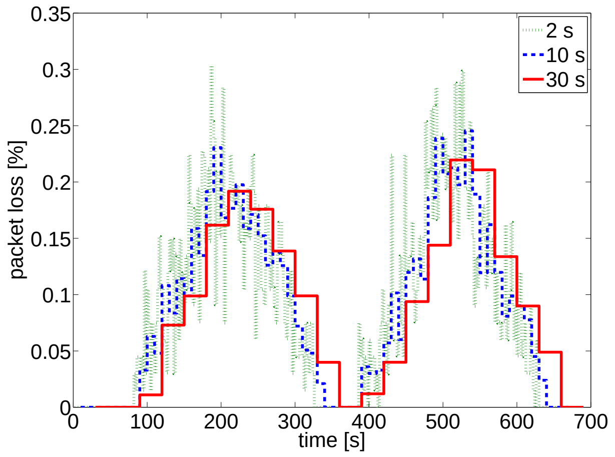


Figure 8: Loss at different resolutions

A includes the IP IDs of all packets sent during the last measurement interval, host B is always able to exactly calculate the current loss rate. Thus, it is of more interest to study the influence of the time between two summary packets. We set up a scenario where we increase the current packet loss by five percent every 30 seconds until it reaches 20 percent, then we decrease it again by five percent every 30 seconds until it is back to zero percent. After 60 seconds we repeat the entire procedure. Figure 8 shows the packet loss as observed by the prototype when sending a summary packet every 2, 10 or 30 seconds respectively. Since packet loss occurred randomly and uncorrelated a measurement interval of 2 seconds leads to heavily fluctuating values as it is too short to capture the actual mean value. While a measurement interval of 30 seconds quite accurately reflects the current loss rate, it reacts too slowly to changes in the network. That is, the end user would have to wait for 30 seconds until the traffic light would adapt to the current circumstances. We therefore chose an interval length of 10 seconds as a good trade-off between accuracy and up-to-dateness.

So far, we have shown in how far our prototype is able to capture the current dynamics in the network. In the following we concentrate on how this translates to user-perceived quality. We again used a VoIP call and varied the properties of the network over time. Figure 9 shows the measured one way delays on the left y-axis and the current color of the traffic light on the right y-axis. We started with a plain connection between host A and host B, translating into a green color. After 60 seconds we increased the one way delay to 50ms, which did not change the color of the traffic light. After another

60 seconds we increased the delay to 150 seconds with a jitter of 30ms and introduced an additional packet loss of 3 percent. After a slight delay the color of the traffic light changed to yellow. When we further increased the delay to 300ms with 40ms of jitter

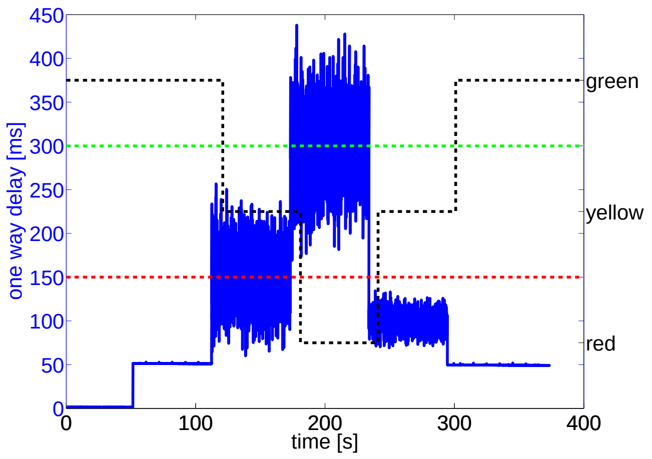


Figure 9: Traffic light and delays

and 10 percent packet loss, the traffic light finally changed to red signaling a bad quality. After 60 seconds we decreased the delay to 100ms with 10ms of jitter and the packet loss to 2 percent, which was reflected by a yellow traffic light. Finally, we returned to 50ms delay without any jitter or packet loss and observed a green traffic light.

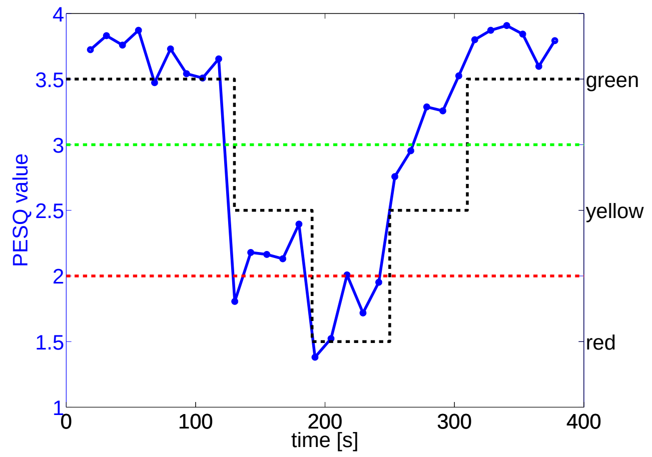


Figure 10: Traffic light and PESQ

To show that the color of the traffic light of our prototype actually reflects user perceived quality, we recorded the voice quality on application level at host B and compared it to the original quality as sent by host A. To do so, we periodically calculated the Perceptual Evaluation Of Speech Quality (PESQ) value [16] of a typical 10 second conversation [17], which was repeatedly transmitted with a 2 second pause between transmissions. A PESQ value above 3 corresponds to good quality, while it is still considered to be acceptable between values of 2 and 3. Figure 10 shows the current PESQ value on the left y-axis and the color of the traffic light on the right y-axis for the previously described scenario. The color of the traffic light corresponds to the trend of the PESQ values. It stays green while the PESQ value is above 3.5 and changes to yellow once the PESQ value decreases to values between 2 and 2.5. During the phase in which the traffic light shows a red color, the PESQ value falls as low as 1.4. Note that the PESQ value constantly rises during the following yellow phase, while the network settings rather imply steady values. The reason is that Skype actually adapted its voice codec to the current condition of the network.

This is also directly reflected by the bandwidth used by Skype. Figure 11 plots the current throughput of Skype against the time of the measurement. At first, we observe that Skype obviously uses some kind of silence detection. It decreases the bandwidth on the uplink of host A during the 2 second pause after each of the 10 second conversation samples. It also reacts to the current condition of the network. Between the measurement time of 200 and 300 seconds it constantly increases the bandwidth to work against the higher delay and packet loss in the network. It is also interesting to see that the

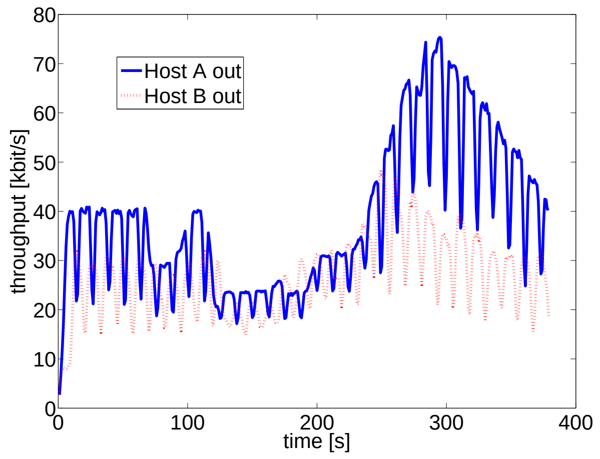


Figure 11: Bandwidth used by Skype

reverse direction, i.e. host B to host A, seems to be correlated to the original direction. During this measurement we did not transmit anything but silence from host B to host A. However, instead of a constant bandwidth, the curve of this link shows a similar shape to that of host A to host B. Apparently, there must be some feedback mechanism incorporated into a Skype VoIP call.

6 Conclusion

The quality of a service as it is perceived by the end-user is an aspect which is often neglected in classic network management. In this paper we presented a self-organizing approach to evaluate the quality of specific end-to-end communications. It measures important network parameters like the one-way-delay, jitter and packet loss on both endpoints of the communication. Correlating these traffic characteristics of both communication partners, it translates the measured values to an easy to understand traffic light scheme. It can either passively monitor the quality of an already ongoing exchange of information or generate predefined traffic patterns to validate whether the current state of the network would support a specific service.

As a proof-of-concept, the algorithm was integrated as an additional module into our already existing distributed network management environment. Using this self-organizing framework, the end-user only has to specify which service should be monitored, as e.g. ftp, VoIP or videoconferencing. The feasibility of this concept was verified by monitoring a Skype VoIP call in a small testbed. It could be shown that the color of the traffic light corresponded to the current PESQ value. That is, the traffic light reflected the subjective impression of the voice quality. The results can be used by network administrators to inspect the current state of their network, by service providers to verify that they preserve a negotiated service level agreement or by (mobile) users to see whether the quality of their current connection allows the use of a specific service.

Acknowledgements

The authors would like to thank Kurt Tutschku and Markus Fiedler for their ideas and the insightful discussions during the course of this work. Special thanks go to Thomas Zinner for the active help during our prototype study.

References

[1] A. Binzenhöfer, K. Tutschku, B. auf dem Graben, M. Fiedler, and P. Arlos, “A p2p-based framework for distributed network management,” in *New Trends in Network Architectures and Services, LNCS 3883*, (Loveno di Menaggio, Como, Italy), January 2006.

[2] P. Maymounkov and D. Mazieres, “Kademlia: A peer-to-peer information system based on the xor metric,” in *IPTPS 2002*, (MIT Faculty Club, Cambridge, MA, USA), March 2002.

[3] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications,” 1996. RFC 1889.

[4] A. Pasztor and D. Veitch, “PC based precision timing without GPS,” in *SIGMETRICS ’02: Proceedings of the 2002 ACM SIGMETRICS international conference on*

Measurement and modeling of computer systems, (New York, NY, USA), pp. 1–10, ACM Press, 2002.

[5] D. L. Mills, “RFC 1305: Network time protocol (version 3) specification, implementation,” Mar. 1992.

[6] D. L. Mills, “A brief history of NTP time: memoirs of an Internet timekeeper,” *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 2, pp. 9–21, 2003.

[7] S. B. Moon, P. Skelly, and D. F. Towsley, “Estimation and removal of clock skew from network delay measurements.,” in *INFOCOM*, pp. 227–234, 1999.

[8] V. E. Paxson, *Measurements and Analysis of End-to-End Internet Dynamics.* PhD dissertation, University of California, Lawrence Berkeley National Laboratory, April 1997.

[9] M. Fiedler, S. Chevul, O. Radtke, K. Tutschku, , and A. Binzenhöfer, “The network utility function: A practicable concept for assessing network impact on distributed services,” in *19th International Teletraffic Congress (ITC19)*, (Beijing, China), 9 2005.

[10] I.-T. R. G.114, “One way transmission time,” May 2003.

[11] I.-T. R. G.107, “The e-model, a computational model for use in transmission planning,” December 1998.

[12] L. Carvalho, E. Mota, R. Aguiar, A. F. Lima, J. N. de Souza, and A. Barreto, “An e-model implementation for speech quality evaluation in voip systems.,” in *10th IEEE Symposium on Computers and Communications (ISCC 2005)*, (Murcia, Cartagena, Spain), pp. 933–938, June 2005.

[13] J. Welch and J. Clark, “A proposed media delivery index.” Internet Draft, August 2005.

[14] Microsoft, “.NET framework, http://msdn.microsoft.com/netframework/.”

[15] M. Carson and D. Santay, “Nist net: A linux-based network emulation tool,” *ACM SIGCOMM Computer Commununications Review*, vol. 33, pp. 111–126, 2003.

[16] ITU-T Recommendation P.862, “Perceptual evaluation of speech quality (PESQ), an objective method for end-to-end speech quality assessment of narrowband telephone networks and speech codecs,” 2001.

[17] Signalogic, “Speech Codec Wav Samples,
http://www.signalogic.com/index.pl?page=codec_samples.”