TraceQoS

A tool for locating QoS failures on an Internet path

Diploma Thesis

written by Chris Welti

Supervisors:

Simon Leinen (SWITCH)
Willi Huber (SWITCH)
Placi Flury (ETH Zürich)
Prof. Dr. Bernhard Plattner (ETH Zürich)

Abstract

English:

Nowadays, recognizing and locating failures of the Quality of Service (QoS) of a network path in the Internet is very difficult, even for network operators. Especially when a problem occurs on a path that is crossing multiple management domains, tracking down the component that is causing the QoS failure is almost impossible.
To address this problem, TraceQoS, a network debugging tool similar to "Traceroute" [28] has been designed and implemented as a prototype. Instead of showing routing information, it provides QoS information[1] on an IP path. This is achieved by running performance tests on the network links between adjacent TraceQoS servers which allows locating network links with poor performance regarding QoS metrics. While one-way measurements are usually not interesting for end-users (they are only interested in overall response time), it is important to know one-way performance for debugging network failures because paths and especially traffic patterns are often asymmetric.
Network operators can install TraceQoS servers on (often already existing) workstations at their POP[2]'s, allowing to test the QoS properties of their network infrastructure.
The prototype has already been successfully deployed on 3 workstations in the network of SWITCH, and more network providers are already interested to participate in tests.

Deutsch:

Das Entdecken und Lokalisieren von Quality of Service (QoS) Fehlern auf Netzwerkpfaden im Internet gestaltet sich heutzutage sehr schwierig, nicht nur für den Endbenutzer, sondern auch für Netzwerkbetreiber. Besonders falls es sich um einen Fehler handelt, der auf einem Netzwerkpfad vorkommt, der mehrere Management Domains umfasst, ist es nicht einfach, die genaue Ursache respektive den genauen Ort des Problems festzustellen.
Um dieses Problem zu lösen, wurde TraceQoS entwickelt, ein Software-Tool zur Lokalisierung von Quality of Service (QoS) Fehlern in Computernetzwerken, ähnlich wie das Programm „Traceroute" [28], das der Aufspürung von Routing Problemen dient.
Anstatt Informationen zum Routing anzuzeigen, wie dies bei Traceroute der Fall ist, werden bei TraceQoS Leistungsinformationen über QoS-Merkmale[1] von Ein-Weg-Netzwerkpfaden ermittelt. Dies erlaubt es, Netzwerk-Links mit ungenügender Leistung bezüglich eines solchen QoS-Merkmals zu lokalisieren. Während Ein-Weg Messungen für den Endbenutzer normalerweise nicht interessant sind, ist es für Netzwerk-Spezialisten wichtig, die Ein-Weg Eigenschaften von Netzwerk-Verbindungen zu wissen, da die einzelnen Verbindungen wie auch die Verkehrscharakteristiken auf den einzelnen Links sehr oft asymmetrisch sind.
Netzwerkbetreiber können TraceQoS Server auf ihren (oftmals bereits bestehenden) Workstations bei ihren POP[2]'s installieren, um es andern zu ermöglichen, die QoS Eigenschaften ihrer Netzwerke zu testen.
Der Prototyp wurde schon erfolgreich auf 3 Workstations im Netzwerk von Switch installiert und mehrere andere Netzwerkbetreiber sind bereits daran interessiert, TraceQoS einzusetzen.

[1] e.g. one-way delay (OWD), one-way delay variation (OWDV), one-way packet loss (OWPL)
[2] POP = "Point of Presence"

CONTENTS

1 Introduction

The Internet is increasingly being used for transmission of real-time generated data, like for example, the exchange of encoded video-streams. Experience shows that the transmission of such signals over the Internet is not without problems, even if it looks like there is more than enough bandwidth available at first. If the dispersion of the transmission times of individual IP packets is too high, video codecs may lose synchronization. In such a case it is extremely difficult for the end-user to find the reason for the lack of transmission quality, respectively the weakest link in the chain of transmission links and active devices along the transmission path. "Ping" and "Traceroute", the most used test programs for detecting network problems nowadays, are not sufficient for that purpose.
Therefore a tool is needed that allows end-users and network administrators to test the quality of a network connection to find and locate errors quickly.
Even network operators experience difficulties in locating QoS failures, especially if different management domains are crossed by the end-to-end path. Often every network operator claims his network is working perfectly and the fault has to be somewhere else.
The TraceQoS tool would allow users to run different types of active quality-of-service measurements along an end-to-end network path. Users and operators could use this tool to determine the probable location of a quality-of-service "fault", much in the same way as they are using Traceroute today to find the location of routing faults.
When the location of a QoS failure is located, the corresponding management domain can be contacted and the failure can be further evaluated within that domain.

2 Related Work

In this section a short introduction to the two most used tools for network debugging, Ping [32] and Traceroute [28] is given. This should help to understand the basics of measurement principles in an IP environment:

2.1 Ping

Ping is one of the basic tools in networking. It is used to test connectivity and measure delay between hosts on an IP network.

Basically an initiating host sends an ICMP "echo request" packet to a destination host which instantly sends back an ICMP "echo reply" packet. The source host measures the time it takes until it receives the "echo reply", which results in a two-way delay measurement (called round trip time (RTT)).

2.2 Traceroute

Traceroute attempts to trace the route an IP packet would follow to some Internet host by sending UDP probe packets or ICMP echo packets with increasing TTL[3] values to the destination and waiting for responses of the routers (hops) between.
The IP protocol requires that each router decreases the TTL of a packet before it is forwarded. When the TTL of an IP packet reaches 0, it is discarded and an ICMP "time exceeded" is sent to the originating address of the packet.
The first probe IP packet that Traceroute sends has a TTL of 1, which is decreased to 0 by the first router on the way. The router usually drops a packet that reaches a TTL of 0 and SHOULD send an ICMP "time exceeded" back to the source IP address. As this ICMP packet also includes the IP address of the router, all routers on the path can be recorded by simply sending probe packets with increasing TTL's until the destination host is reached.
Unfortunately no "time exceeded" packet is generated when the packet arrives at the destination. So how can be determined when the destination is reached? Depends on whether UDP or ICMP probes are used.
For ICMP probes the idea is to send ICMP "echo request" packets. It is specified in the IP protocol that such packets have to be answered by the destination host with an ICMP "echo reply" packet. As soon as an ICMP "echo reply" is received, it implies that the destination host was reached. Unfortunately, when Traceroute was developed, a lot of routers didn't send "time exceeded" ICMP packets for ICMP "echo request" packets, so no response came from the routers....
This is why UDP probes were used in the first implementation. (Nowadays, routers usually do send "time exceeded" messages for ICMP probes).
Sometimes, Traceroute is also used for performance measurement (RTT on sub-paths) and ICMP packets might get preferred treatment at routers, thus to get more "realistic" results, using UDP packets seems to be the way to go.
To find out when a UDP probe has reached the destination is rather difficult: Up on arrival of the UDP probe at the destination machine, no ICMP "echo reply" message is sent back.
Luckily, with a little trick, an answer can be forced from the destination host without running special software on it. If an Internet host receives an IP packet with a destination port number that is not in use[4] on the destination host, an ICMP "port unreachable" message is returned to

[3] TTL = “Time to Live”, actually a hop-count limit.
[4] No server is listening for packets arriving at this port.

the sender. As this is specified in the IP protocol, each host that runs an IP implementation should follow that procedure.
Thus, for UDP probes, the destination port number is set to an unlikely value. Unfortunately, if that port is actually being listened to by the destination host for another purpose, the packet will usually just be dropped and no ICMP message is returned. Although this is highly unlikely, Traceroute implementations usually allow you to set the initial port number manually.

To reduce waiting time, modern Traceroute implementations also use the fast-spray mode which sends a couple of probe packets in parallel. A complete trace is then available almost immediately, within the slowest RTT of all packets sent.

2.3 The downside of ICMP probing

Unfortunately, malicious use of the ICMP[5] has led to mechanisms that restrict the efficacy of ICMP probing. Several host operating systems (e.g. Solaris) now limit the rate of ICMP responses, thereby artificially inflating the packet loss rate reported by ping. For the same reasons many networks (e.g. microsoft.com) filter ICMP packets altogether. Some firewalls and load balancers respond to ICMP requests on behalf of the hosts they represent, a practice that is called ICMP spoofing, thereby precluding real end-to-end measurements. Finally, at least one network has started to rate limit all ICMP traffic traversing it. []

2.4 Professional Tools

For an overview of professional tools available for traffic measurements, see [5].
Two of the most widely spread systems to measure one-way delays are Surveyor [9] and RIPE (TTM) [34]. However these two projects are infrastructures that perform tests on regular intervals rather than on demand. Additionally they require special hardware and GPS.

[5] Mainly DoS attacks (Denial of Service) that flood a certain host with ICMP packets.

3 Specification

In this section it is elaborated what exactly is expected to be achieved by the TraceQoS tool.

3.1 Field Of Application

As already indicated in the introduction, TraceQoS is not intended for deployment by end users, but by network operators. They install TraceQoS measurement stations, so called "TraceQoS Servers", at their border routers or other points of interest in their network topology.

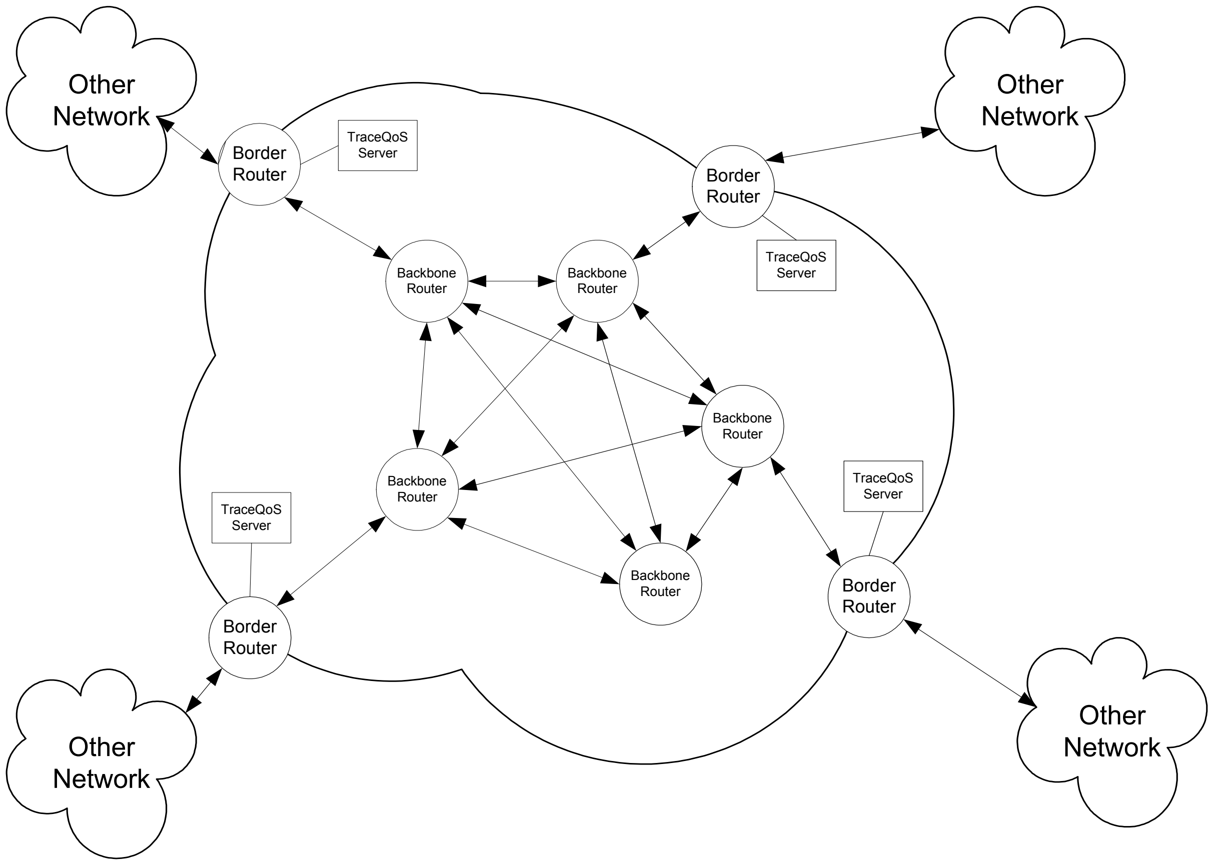


Figure 1: Deployment of TraceQoS servers

3.2 Goal

The goal of TraceQoS is to locate "failures" of Quality of Service (QoS) along a given path of the Internet. It will not be able to reflect exact information about QoS parameters in a well-functioning network, but it should allow narrowing down a particular failure to the area where it evolves. In other words TraceQoS should be able to find out which part of a network path is responsible for a failure of QoS. To do this, it is expected from the TraceQoS tool to show QoS metrics of a network path between a source host and a destination host. Instead of investigating only the end-to-end connection, all links on the path between the source and the destination shall be measured separately with test traffic. Unlike many currently available measurement tools, TraceQoS measurements will be performed one-way (unidirectional), as

in the context of this work, the interest lies in particular link behaviour and not in end-to-end performance.[6]
See also Appendix B for a further explanation why one-way measurements are required.

3.3 Definition of Quality of Service (QoS)

Quality of Service is a very vague term that is interpreted differently by almost everyone that encounters and uses it. In the context of this project, we refer to Quality of Service as an indication of the “quality” of a network path (=”service”).
In the current Internet, the most important metrics for the quality of a network service seem to be packet delay, packet delay variation and packet loss.[7] Although available bandwidth is also considered to be important by some individuals, it is not requested in the context of this project, as there is still a discussion going on what exactly available bandwidth means and how it could be measured.

3.3.1 Definition of a QoS failure

In this document, a failure of QoS occurs if one or more QoS parameters are out of its “usual” range. Although this definition may seem very vague, the notion of when a QoS failure occurs is entirely up to the user. It is important to see that failure of QoS is tightly associated with the application that uses the network.
TraceQoS is not supposed to show reliable values of network performance when the network is well functioning, rather it is supposed to be utilized only after a failure of QoS is suspected. It is not supposed to detect QoS failures by itself but merely help users and operators to locate and isolate problem zones wherein QoS failures occur.
TraceQoS is going to be an active measurement tool that will inject test packets into the network and evaluates how the packets are being delivered by the network.

3.4 QoS Measurements

The QoS metrics to be investigated by the TraceQoS tool include: delay, delay variation and packet loss. To gather data about those metrics, probe packets are sent in a user configurable manner. The kind and number of probe packets to send to make a reasonable statement about the investigated link depends on which kind of traffic pattern one would like to approximate. Therefore such parameters can not be pre-defined, and should be specified by the user of the TraceQoS tool. Note that measurement results are always connected to the type of packets used for tests. Therefore test results should be reported along with an indication of the type of packets used for test.

3.4.1 Definition of the singleton measurement metrics

The singleton QoS metrics to be used for TraceQoS are one-way delay, one-way delay variation and one-way packet loss. All other metrics used, will be deduced from these singleton metrics.

3.4.1.1 One-way delay (OWD)

With t_sent being the time when the first bit of an IP packet left the src and t_rcvd being the time when the last bit of the IP packet arrived at the dst, one-way delay is defined as the difference t_rcvd - t_sent [compare RFC 2679 section 3.4.].

[6] This way a failure can be narrowed down to the links investigated.
[7] By no way is this meant as a complete list; it is the impression of the author after reading through many newsgroups, papers and other articles that those parameters are generally considered the most useful.

As a pre-requisite for one-way delay measurements we have to assume that time synchronisation between participating hosts is provided.
For TraceQoS, a one-way delay accuracy in milliseconds in the one-digit range is desired. In this project that range is considered sufficient to show up "QoS failure" on investigated links.[8]

3.4.1.2 One-way packet loss (OWPL)

Packet loss occurs when a packet sent from the source host to the destination host is not received by the destination host within an appropriate time. [compare RFC 2680 section 2.4.].

3.4.1.3 One-way delay variation (OWDV)

For a singleton delay variation value, two IP packets with one-way delays d1 and d2 are compared. The singleton delay variation dv is defined as the difference between d1 and d2. dv = |d2-d1|. If one of the two OWD values is undefined, OWDV is also undefined.
Usually for OWDV measurements, OWDV is only computed between IP packets that have the same size. (As it takes a longer time to transmit large packets than small packets, subsequent IP packets with different packet sizes naturally have different OWD's, but that has nothing to do with network performance). Note that for measurements of OWDV, time synchronisation is not required, as only differences in times of the same base are considered[9].
Delay variation is often more critical to applications than delay (e.g. for video broadcast).

3.4.2 Measurement methodology

This section gives an overview on how the QoS parameters described in 3.4.1 should be measured between two hosts. It is based on RFC2679 [26], Section 3.6.

- Arrange that the source and destination host are synchronized; that is, that they have clocks that are very closely synchronized with each other and each fairly close to the actual time.

- At the source host, select source and destination IP addresses, and form a test packet with these addresses. Any 'padding' portion of the packet needed only to make the test packet a given size should be filled with randomized bits to avoid a situation in which the measured delay is lower than it would otherwise be due to compression techniques along the path.

- At the destination host, arrange to receive the packet.

- At the source host, place a timestamp in the prepared test packet, and send it towards the destination host.

- If the packet arrives within a reasonable[10] period of time, take a timestamp as soon as possible upon the receipt of the packet.
  By subtracting the two timestamps, an estimate of the one-way delay can be computed. Error analysis of the implementation of the method must take into account the closeness of synchronization between the source host and destination host, if it is known. If the delay introduced by additional links from the router to its associated TraceQoS server is known, then the estimate could be adjusted by subtracting this

[8] While there certainly are applications that need a more precise resolution of OWD measurements, in the scope of this project the proposed range seems appropriate.
[9] For OWDV, single OWD measurements are compared. As these are all composed under almost the same conditions, time differences between hosts do not matter as long their clocks are not drifting apart.
[10] Note that the threshold of 'reasonable' is a parameter of the implementation.

amount; uncertainty in this value must be taken into account in error analysis. Furthermore, if the delay between source host's timestamp and the actual sending of the packet is known, then the estimate could be adjusted by subtracting this amount; uncertainty in this value must be taken into account in error analysis. Similarly, if the delay between the actual receipt of the packet and destination host's timestamp is known, then the estimate could be adjusted by subtracting this amount; uncertainty in this value must be taken into account in error analysis[11].

- If the packet fails to arrive within a reasonable[10] period of time, the one-way delay is taken to be undefined (informally, infinite). The packet is then considered as lost.

To measure the metrics we are interested in, it is sufficient to send test packets that include the timestamps of the time of sending (needed for OWD, OWDV) and a sequence number (needed for OWPL).
However keep in mind that measurements depend on the type of packets used, because packet parameters as type (like UDP, TCP) and packet size influence how the components along the path (hosts, network) treat those packets. For this reason the type of packet used should also be presented along with measurement results.

3.4.3 Accuracy

Without the use of locally attached GPS devices for synchronizing time, accuracy for OWD measurements is targeted at the low 2-digit millisecond range, for reasons of difficulties with synchronizing clocks with using NTP. However this should be sufficient to identify problematic links across a path.
For OWDV accuracy lower than 1 millisecond can be expected, as it does not depend on synchronized clocks. For OWPL all packets sent will be accounted for.
As most communication in the Internet is based on IP packets, the scope of TraceQoS is limited to network behaviour of IP traffic. For an example of an investigated IP-path see Figure 1.

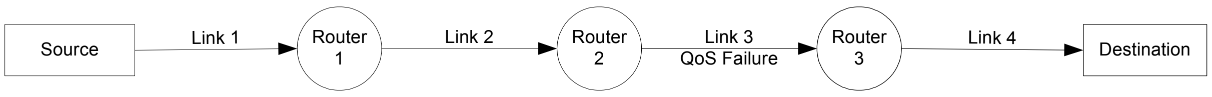


Figure 2: IP path over 3 routers

3.5 TraceQoS tasks

The following tasks are to be executed by a TraceQoS tool:

- **Find all routers on the way from the source host to the destination host**
- **Map the router path to a sequence of TraceQoS probes**
  A mapping from the router path to a sequence of TraceQoS probes, where the concatenation of the paths between subsequent probes represents a good approximation of the normal network path from source to destination has to be found.
- **Perform active one-way measurements between two TraceQoS probes.**
  As no stand-alone tool is freely available to conduct host-to-host measurements of OWD, OWDV and OWPL, such a component has to be developed.

[11] As it is not the goal of TraceQoS to be as accurate as possible, no error analysis/correction is currently planned.

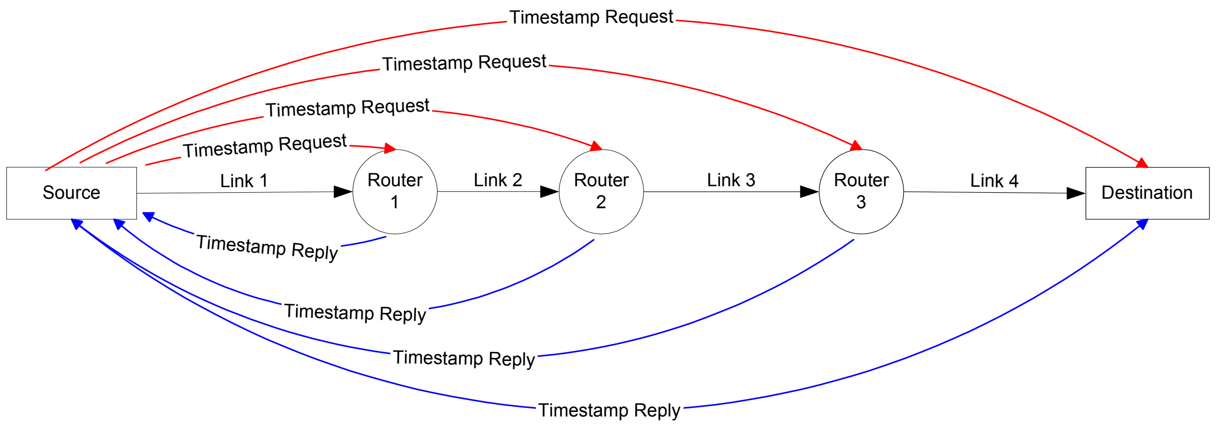
- **Perform such measurements automatically between adjacent TraceQoS servers over the end-to-end path**
  A protocol has to be developed that allows such measurements to be automatically initiated between all adjacent TraceQoS probes and the results to be collected at the host that manages a TraceQoS session.

- **Display the results of the measurements.**
  Results should be gathered and displayed at the machine that coordinates the measurements.

4 Possible design solutions

For one-way delay and one-way delay variation measurements it is basically sufficient to have timestamps of when the packets have been sent and when they were received. To detect packet losses, each packet has to be fitted with sequence numbers.
Other measurements are not currently planned for implementation, and might need special precautions to work in the TraceQoS model. (e.g. measurements that include acknowledgements in the reverse direction)

4.1 Using ICMP packets



From the source host, send ICMP "timestamp request" packets to each router on the path from Source to Destination. Although this path may not be static, a “Traceroute” [28] is performed at the beginning to find the routers. For simplicity, it is just assumed all packets from Source to Destination will take that path.
Then, collect ICMP "timestamp reply" packets, and also record a timestamp at arrival of those packets, so performance for links in both directions can be calculated.

e.g. (Source, Router 1), (Source, Router 2), (Source, Router 3), (Source, Destination)

Of course this is not true hop-to-hop link performance that is measured, but it may be approximated by subtracting preceding (Source, Router) measurement.

e.g. (Router 1, Router 2) = (Source, Router 2) - (Source, Router 1)

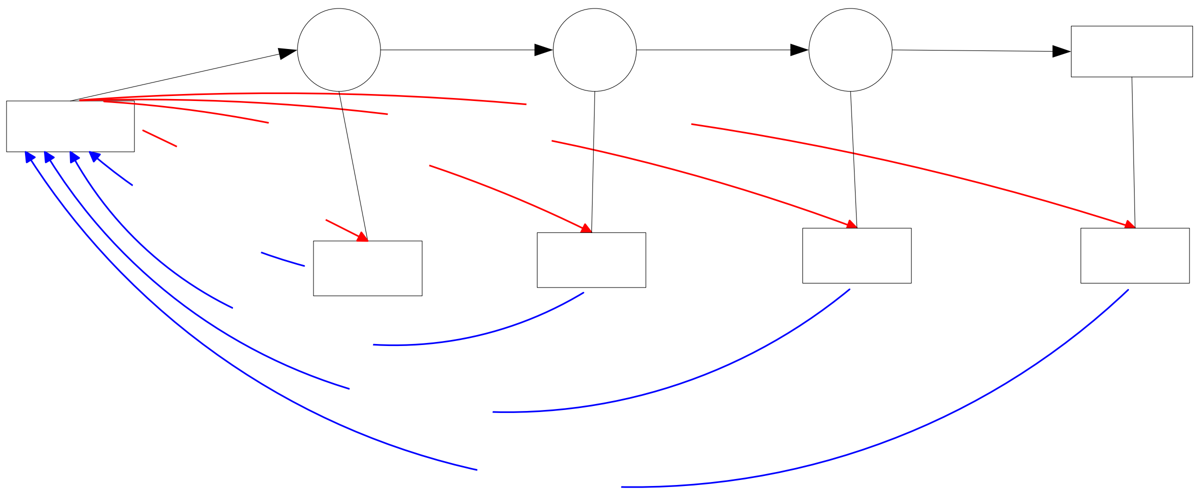
Unfortunately, it is possible that the path to the destination changes, while TraceQoS is running, which could lead to misleading results.
(e.g. (Source, Router 2) is not routed as (Source, Router 1, Router 2), but instead through an other router Router X (Source, Router X, Router 2))

Possible metrics:	OWD, OWDV, OWPL
Impossible metrics:	Bandwidth, etc.
Advantages:	Simple to implement, no modifications needed on routers (since it is based on ICMP which is already part of every IP router anyway), works out of the box
Drawbacks:	Unfortunately, ICMP "timestamp request" packet is not being processed by hardly any IP implementations, so often the ICMP packet is just

discarded by the router and no ICMP "timestamp reply" is sent. Furthermore ICMP traffic is often filtered or traffic controlled. In case rate control for ICMP packets is applied, measurements would not reflect "normal" traffic characteristics. See also 2.3.

4.2 Using TraceQoS servers

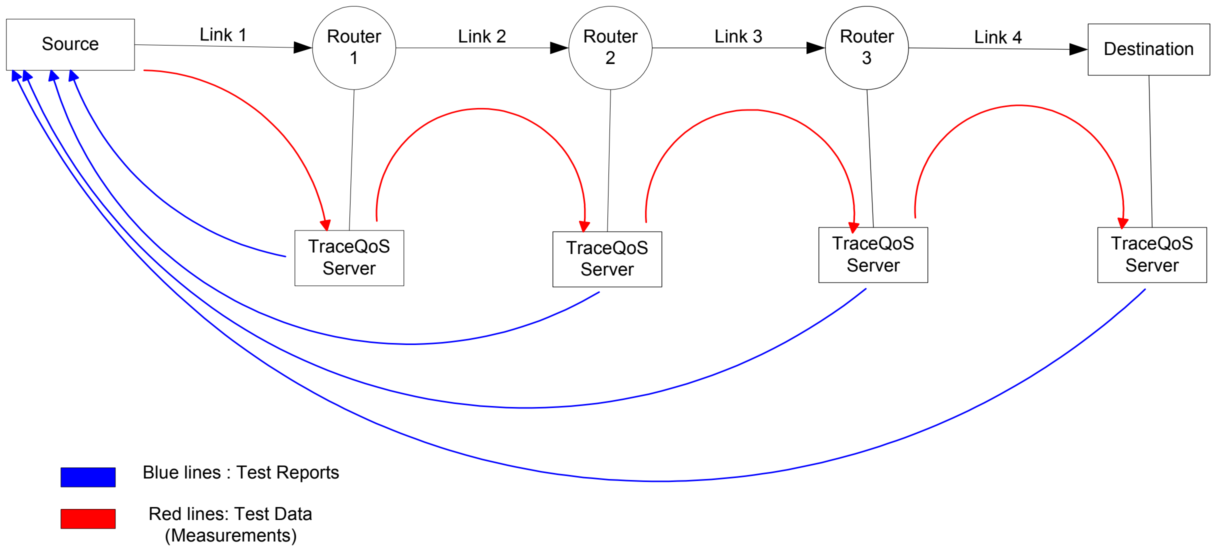
Basically the same approach as in 4.1 except that each router on the way needs to have an associated server responding to "performance test requests". As routers are usually hardware devices that do not allow the installation of additional customized software, performance tests are performed via a host as close to the router as possible. That implies that a directory of TraceQoS servers has to be kept, so that a mapping from a router to the associated (or closest) TraceQoS server can be performed. Also this reduces availability of suitable probing hosts considerably, as not each router in between will have an associated server.



Possible metrics:	OWD, OWDV, OWPL, bandwidth etc.
Impossible metrics:	None
Advantages:	All kind of metrics could be measured depending on the implementation, since it is a custom application at both ends
Drawbacks:	Performance measurements lose in accuracy since they are not directly between host and router, but host-router-host. Additionally, if the route changes, measurements may be misleading as performance is always measured between the source and the TraceQoS server, and not between links. A lot more difficult to implement than 1). Does not work out of the box, needs agents installed on hosts very close to each of the routers. Also needs some sort of TraceQoS server discovery.

4.3 Performing link tests between TraceQoS servers

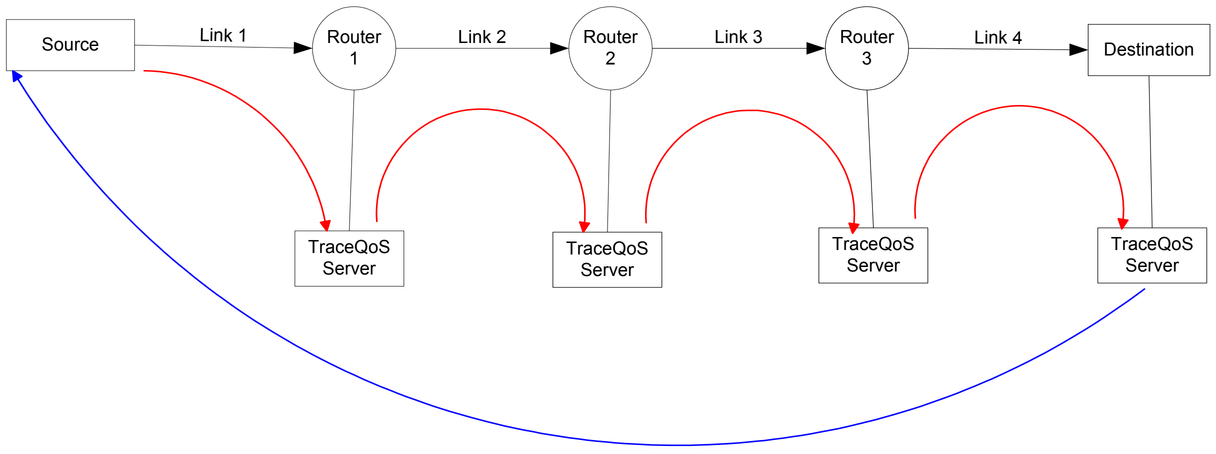
In approaches 4.1 and 4.2, measurements are always made between the source host and a remote host. As already mentioned this leads to unreliable results. Measuring directly between adjacent TraceQoS servers should provide more accurate and reliable results for link performances.
This could be done in the following manner:
Given the route (Source, Router 1, Router 2, Router 3, Destination), measurements will be performed as follows:
Source sends a link test request to Router 1.
After Router 1 has received the link test request from Source, it sends an acknowledgement and gets ready for the performance measurement. Source will then initiate the particular performance measurement and after it is done, it will collect the results from Router 1. After completion, Router 1 will initiate a link test request to the next router in the path (Router 2 in this example). Same routine as before, only that Router 2 will send the test results directly back to Source, which will be collecting all test results.
Then Router 2 initiates a test session to Destination, and Destination forwards the test results back to Source.
Note: After each measurement, the result has to be returned to the initiating host (Source), which will generate an output with all the accumulated data.
Again, instead of the routers themselves, hosts nearby have to be used.



Possible metrics:	OWD, OWDV, OWPL, bandwidth etc.
Impossible metrics:	None
Advantages:	All kind of metrics could be measured depending on the implementation, since it is a custom application at both ends. Measurements are pretty accurate as they are measured link-2-link.
Drawbacks:	Performance measurements lose in accuracy since they are not directly between host and router (respectively router-router), but host-router-host. Most difficult to implement. Does not work out of the box, needs TraceQoS servers installed on hosts very close to each of the routers. Also needs some sort of TraceQoS server discovery.

4.4 Each packet time-stamped by all TraceQoS servers

In an approach similar to the IPMP proposal for routers, the idea is to send a stream of packets on the way from source to destination via all the TraceQoS servers which add an entry with the timestamp of receipt and path information (their IP address, the associated router IP address, time accuracy (method of synchronisation), time to router (ping)) to each packet and send forward them to the next TraceQoS server on the way.



🟦 Blue line: Packets with timestamps of all TraceQoS Servers

🟥 Red lines: Timestamped packets being forwarded

Possible metrics:	one-way delay (OWD), one-way jitter (OWDV), one-way packet loss (OWPL) (unfortunately you can not tell on which link a packet gets lost... all information about the route the lost packet took is lost),.Basically everything is possible that may be calculated with only the timestamps at hand
Impossible metrics:	everything that requires more than timestamps
Advantages:	Measurements are pretty accurate as every TraceQoS server on the route adds a timestamp. Furthermore, traffic is reduced as only one packet is sent for the entire route. All data analysis is performed at the TraceQoS manager.
Drawbacks:	Performance measurements lose in accuracy since they are not directly between host and router (respectively router-router), but host-router-host. Measurements results are not visible until Destination is reached and reports back to Source. If there is a failure on the path, you can not discover where it was (and that's the whole point of measuring) Difficult to implement. Does not work out of the box, needs TraceQoS servers installed on hosts very close to each of the routers. Also needs some sort of TraceQoS server discovery.

5 Design Proposal

In this thesis the approach described in 4.3 is used as it seems to offer the best functionality and extensibility.

5.1 TraceQoS model

What we would like to measure with TraceQoS is the segment-wise performance on the links between a source host and a destination host in just one direction (Later on, a reverse one-way measurement could be easily integrated).

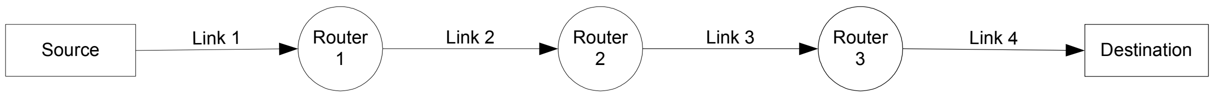


Figure 3: IP path over 3 routers

What we basically want to do is to measure performance between
Source and Router 1,
Router 1 and Router 2,
Router 3 and Router 4,
…
Router n and Destination.
n is the number of routers on a given path.

Figure 3 displays an example of a source host connected over 3 routers to a destination host.

Unfortunately, we can not expect cooperation from routers on the way, as they most certainly can not participate in any kind of requests other than basic IP packet forwarding and maybe some ICMP-message support[12]. As discussed in 2.3 and 4.1, the currently deployed ICMP support is not sufficient to support a measurement system for one-way measurements.
This leads to the solution to attach measurement stations running our TraceQoS server software to routers on the path. Those will then act as substitutes for the routers, in link-to-link measurements. Ideally every router of the Internet would have a TraceQoS server attached.
Unfortunately this would be heavily expensive and is of course totally unrealistic.
In reality TraceQoS is only being planned for deployment at border routers (that connect networks of adjacent network providers) or other particular areas of interest.

A simple example of an IP path from source to destination over 3 hops could look like this:

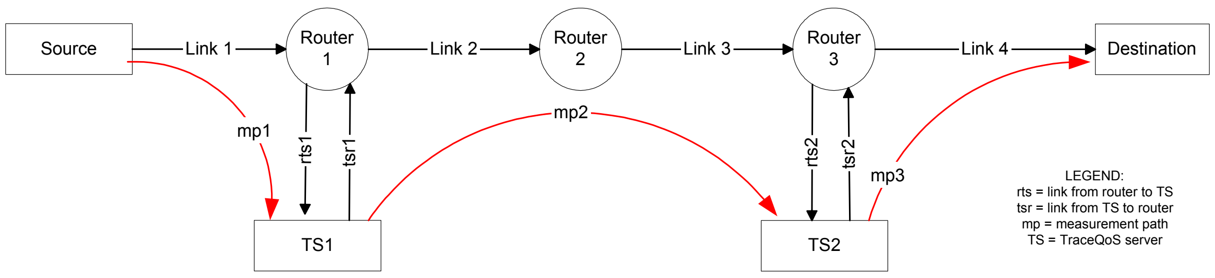


Figure 4: TraceQoS measurement over 3 hops

[12] There are currently some people involved in developing a protocol that assumes routers participate in measurements. See [38]

To investigate the network path from Source to Destination, TraceQoS measurements in the example above should be:

Measurement path 1 (mp1):	Source->Router 1->TS1
Measurement path 2 (mp2):	TS1->Router 1->Router 2->Router 3->TS2
Measurement path 3 (mp3):	TS2->Router 3->Destination

5.1.1 Considerations for the measurement paths

The measurement path between the TraceQoS servers must cover the "normal" sub-path a packet would take. For the example above, measurement between TS1 and TS2 must include the sub-path mp2, Router 1->Router 2->Router 3.
Imagine TS1 being connected to another gateway Router 4 that has a direct link with Router 2.

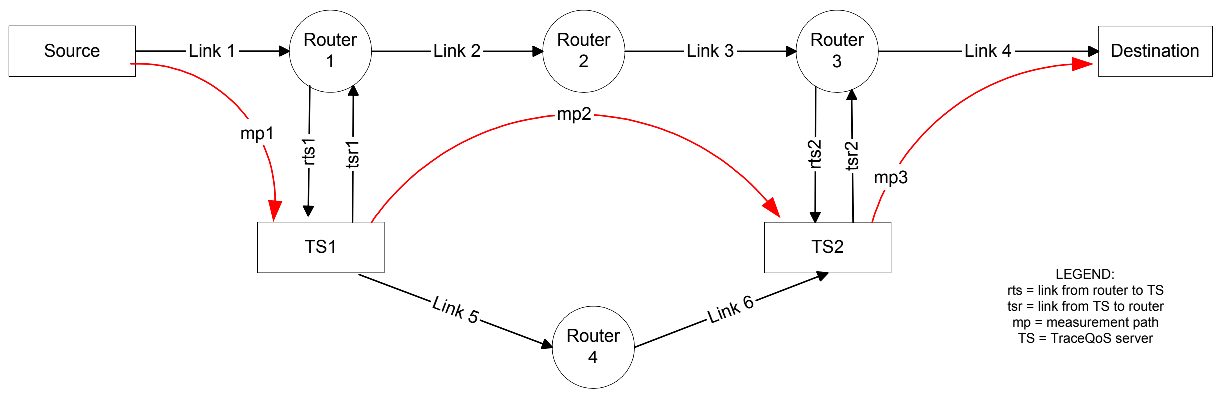


Figure 5: Conserving of measurement path

If Router 4 is configured as the default gateway for TS1, all outgoing IP traffic will pass Router 4. So, when TS1 sends a packet to TS2, the packet would go along the path TS1->Router 4->TS2, which does not include the native sub-path Router 1->Router 2->Router 3. That's something we definitely don't want to happen!
Thus, for any given destination, TS1 must use Router 1 as a gateway (this implies that the default gateway for a TraceQoS server has to be the associated router).
Alternatively, Source Routing option could be used to pre-define the path the packet takes. Since support for Source Routing has steadily decreased due to security reasons (it is feared to be used for man in the middle attacks (spoofing), so it is usually recommended to disabling source routing [29, 30]), this is currently not considered for implementation.

Measurements between the TraceQoS servers (in the example measurements on the paths mp1, mp2 and mp3) should occur simultaneously (in parallel) to reflect a “normal” traffic flow from source to destination. If measurements are taken sequentially, there could be the chance that we miss changes in network traffic/QoS on the links.
For example, when measuring on path mp1, there could be a temporal failure on path mp2 or mp3 that is not present afterwards when those measurements are carried out.

5.1.2 Considerations for IPv4 and IPv6 compatibility

Both IPv4 and IPv6 should be supported. Either a fully qualified domain name or an IP address of the target interface of the destination host has to be provided at the start of TraceQoS. As a FQDN does not usually indicate which IP Version is to be used, an additional flag for forcing a particular protocol address family is needed when using FQDN.
For actual test traffic the IP Version used should always stay the same as it would be the case for an end-to-end connection between the source and destination host. Therefore a flag to force a specific address family for actual test traffic should be used.

5.1.3 Need for a unique Test ID

If more than one TraceQoS session is to happen, the need for a unique test ID emerges to separate different TraceQoS test sessions from each other. As in the TraceQoS model no central component is overlooking all test sessions is present, different TraceQoS sessions are not aware of other sessions in progress. While this is nice because there is no dependency on a central component, which would handle test registration, independently selecting a unique ID for each test is definitely a problem. A reasonable solution seems to be to use a timestamp with microsecond resolution of the time when the TraceQoS process is started as a unique ID for each test. Chances that two or more TraceQoS processes start at exactly the same time[13] are extremely small. This approach is also used in the OWDP Implementation of the Surveyor project [4].
This unique Test ID is not only used to identify the whole test session but also to identify the single tests carried out between two adjacent TraceQoS servers. As the Test ID is unique and there can only be one test traffic flow with the same Test ID between two adjacent TraceQoS servers, there can be no collision of this Test ID at a single host.

5.1.4 Considerations for parallel test sessions

Concurrent tests should not interfere with each other, neither in test setup nor in test traffic.
If more than one test is running on the same link, test results will just be a reflection of current link behaviour. This naturally includes also effects introduced by the test traffic from other sessions. The number of concurrent tests, the bandwidth used by all test traffic should be able to be controlled by configuration options at the TraceQoS servers.
Concurrent test traffic receiving at the same TraceQoS server need to use different server ports, as two different processes can not use the same port at the same time. This requires a port selection at the test beginning, which can be done randomly at the beginning.
Alternatively, if only one port is to be used as a designated “TraceQoS” port[14], a de-multiplexer is needed that handles all incoming traffic according to their locally registered Test ID.

5.2 Implications of the TraceQoS model

5.2.1 Delay measurement

Measuring between TraceQoS servers instead of directly between routers introduces additional delays in the routing path so that the accuracy of results does suffer a bit. Although, in practice, measuring directly between routers would probably not yield better results, as routers are optimised for the “simple” task of forwarding packets[15], and all other tasks are

[13] Respectively the same timestamp. As an exact time synchronisation is not possible, two processes not started on the same machine but at the same time, do not have the exact same notion of current time.
[14] This then should be an IANA assigned port used for TraceQoS traffic, alike for example port 21 for FTP.
[15] Storing and forwarding packets for most routers to be exact.

performed by a “slow” CPU that also has other things to do than replying to echo request packets.
One solution is to install the TraceQoS servers as directly to a router as possible, so no relevant additional network delay is introduced in the measured path.
In the simple case of unadjusted measurements, they would only make sense as long as the delays on the "real" links are a lot higher than on the links between the routers and the TraceQoS servers (delay(Source->Router 1)>>delay(Router 1->TS1)).

As the TraceQoS tool is primarily thought as a debugging tool rather than a measurement tool, introducing negligible additional delays does not really impact its usefulness. Naturally this statement only holds as long as the additionally introduced path does not have a performance problem by itself, which would implicate a "virtual" problem zone not even on the path to be investigated!

However, we could even compensate for the delay introduced by the additional links to the TraceQoS servers. The estimated average delay to the router that they are attached to can be subtracted from measurements. (Remember: what we would like to measure is e.g. delay(Source->Router 1). Using the method above: delay(Source->Router 1->TS1)-delay(Router 1->TS1) ≈delay(Source->Router 1).

This only works as long as the delay from the TraceQoS server to the associated router is approximately symmetric. For the example above this means that delay (TS1->Router 1) ≈ delay (Router 1->TS1) ≈ delay(TS1->Router 1->TS1) / 2 (RTT), which can be measured quite accurately using a simple ping command. Note however, that a ping does not really reflect the delay of the path between Router and the associated TraceQoS server, as the processing time in the router probably is longer than the link delay.

If the TraceQoS server is connected with a full duplex link to the associated router, the condition of having symmetric delays is most certainly true.

Again, remember, it is not the goal to have scientifically accurate measurements, but to get hints at where to find performance problems on a network path. So, what we don't want is having a problem on the network link between the TraceQoS server and the associated router, which would destroy the main purpose of TraceQoS to find QoS network failures on the main path.

For this reason it is encouraged to use error corrections, if and only if they are known.

5.2.2 Delay variation measurements

As long as the delay measurements are valid as described in the paragraph before, there are no special implications for the delay variation measurements as these are simply deducted by the delays. This holds because if there is no negligible additional delay[16] on the link between the TraceQoS server and the associated router, variations of this practically "non-existing" delay does not interfere with measurements.

5.2.3 Packet loss measurements

To analyse what the additional path implies for packet loss, we have to investigate where packet loss happens. Packet loss can be for various reasons:

[16] In particular, queuing delay.

1.) Congestion of queues on routers
2.) Congestion of queues on lower level devices (ATM/Ethernet switches)
3.) Rate limiters
4.) Traffic shaper
5.) Firewall
6.) Transmission errors (due to link misconfiguration, dispersion, attenuation, …) resulting in corrupt packets[17].

To not introduce additional packet loss, we require that a TraceQoS server has to be connected to the associated router in a way that does not introduce any new possibilities of packet loss into the measurement path. Otherwise we could never determine if the packet loss occurred on the original measurement path or on the link from the TraceQoS server to the associated router.

This means no additional possibly congested paths should be added to the path, no rate limiters, no traffic shapers, no firewalls in between the TraceQoS server and the associated router.
To allow simultaneous measurements from and to the TraceQoS server, a properly configured full-duplex link is required in between.

To sum up, TraceQoS servers should be attached over as few unshared links to the associated router as possible to make sure that only negligible packet loss may occur on the connection between them.

5.3 Deduced requirements for TraceQoS servers and their location

- Clocks on the TraceQoS servers must be synchronized accurately.
  At the time of writing time accuracy in the lower one-digit milliseconds range is desired. This should be possible to implement with NTP synchronization (see [16])
  This is mandatory for all one-way delay measurements. (However it is not required for delay variation and packet loss measurements)
- The clock resolution[18] on a TraceQoS server has to be at least 1 ms.
- A TraceQoS server must be connected to the associated router as directly as possible. A direct, full-duplex link[19] to the router is preferred.
- A TraceQoS server must use the associated router as a default gateway.
  (This is to ensure that "real" measurement sub-path is followed)
- If time stamping of packets is done at user-level, there should be no other resource-consuming applications running on that machine, as they might interfere with the process[20]. At heavy load this could introduce imprecise time stamping and introduce artificial packet loss. (See Appendix E.2)
- The network connection between the TraceQoS server and the associated router should not be a bottleneck for the link to be measured (in respect of the metrics to be measured)

[17] Corrupt packets are usually discarded if corruption is detected.
[18] The clock resolution indicates how often the internal clock is updated by the kernel. The more precise the better; the 1ms resolution is chosen because that's about the best accuracy we will get with NTP synchronisation anyway. For GPS synchronisation a clock resolution of around 20 us is desirable)
[19] The link should only be lightly loaded.
[20] For Linux and BSD systems, the socket option SO_TIMESTAMP allows to have kernel timestamps for arrival times of UDP packets.

- Delay from the TraceQoS server to the associated router should be symmetric. (so a OWD can be "safely" approximated with RTT/2).
- There should be no considerable delay jitter from the TraceQoS server to the associated router, as this could adversely affect measurement results. Under the condition that the TraceQoS server is directly connected (or at least very “close”) to the associated router and there is negligible queuing, it can be said that this holds as well.

The requirement that the TraceQoS server has to be located very close to the associated router is actually not that unrealistic. The targeted field of application is to attach TraceQoS servers to border routers of network providers. As many network providers (e.g. SWITCH, GEANT, Abilene) already have workstations installed at their PoP’s (Point of Presence) anyway, the TraceQoS server software could be easily deployed on them.

6 The TraceQoS process

As an example let’s look at the following set-up:

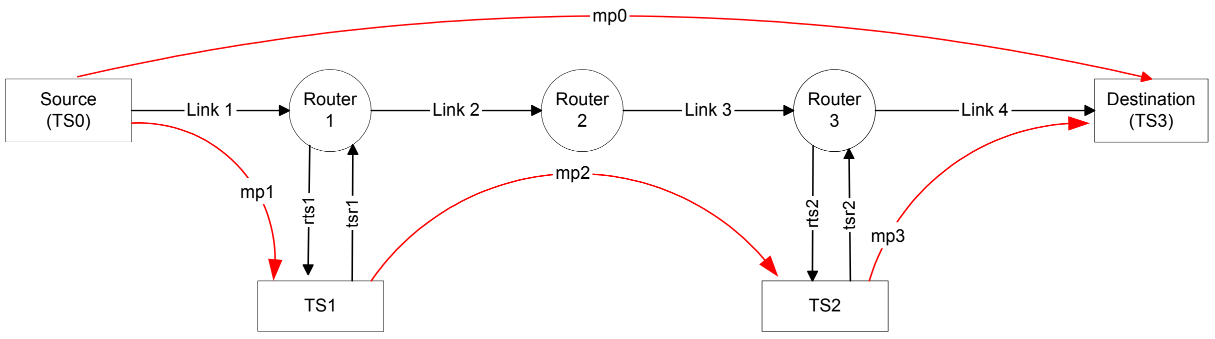


Figure 6: A TraceQoS setup with 3 routers and 2 associated TraceQoS servers

6.1 Test Setup

The source initiates the whole TraceQoS process by starting the TraceQoS manager (TM) that seeks to investigate the path from source to destination.
First the TM has to find out which routers are on the path to be measured. For simplification, it is assumed that the path for typical IP traffic from source to destination would stay the same for the duration of the measurements.
Of these routers the TM has to find associated TraceQoS servers, if any. An elaboration of how this is done follows later on.
All those associated TraceQoS servers found now form the measurement stations between which measurements are going to be made[21]. In the example there is the measurement path mp1 between Source and TS1, mp2 between TS1 and TS2, as well as mp3 between TS2 and Destination. For each measurement path, a separate test session request has to be sent to the sender of a test connection. As a test session always includes a sender and a receiver, a test itself is negotiated between the sender and receiver.
The question now is if the source will make all test setups between the TraceQoS servers, or if test requests are “daisy-chained” along the path. Keep in mind that this only about the setup of tests, not about the flow of test traffic itself.

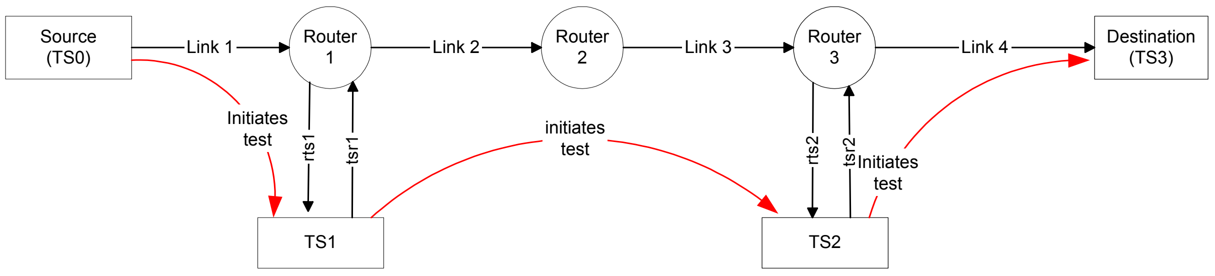


Figure 7: Daisy chained test setup

The daisy chaining approach has the advantage that it is more “natural” to the original measurement path, as naturally packets can only start to flow on a subsequent sub-path after

[21] Note that the TM itself that runs on the client machine does not have to be part of those servers. Although it does improve the significance of the results if more TraceQoS servers are involved it is not a requirement that the Source and Destination participate in actual test traffic.

data has already passed the previous sub-path. Synchronisation for starting the measurements simultaneously is not needed in either case, as the flow of data would be sequential anyway. In case where the source controls all measurement, it has an explicit view of all connections going on, which is important for data collecting.
A first approach for data collecting was the idea that every participating TS would send the results to the source which would be listening for result packets on a specific port for a specific test. Unfortunately that is usually not practicable in real life, as currently firewalls, IDS, or other filters usually deny traffic that was not initiated from within the own network. Thus, measurement results have to be requested by the source itself.
The source needs to know when and where it can fetch the test data from the participating servers. In a daisy chained approach, that information would have to flow back to the source which would complicate the protocol.
If the source initiates each test, it knows, when a test has successfully started, and it can then fetch the results at the corresponding server. If a test fails to initiate, the TM can accommodate a “replacement” test.

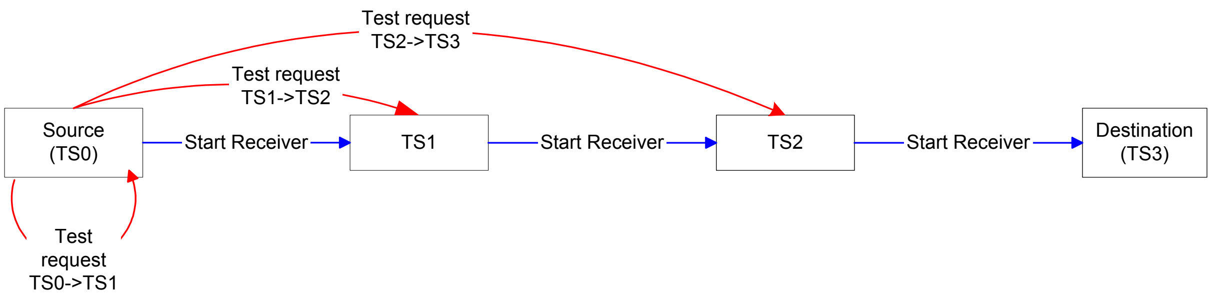


Figure 8: Centralized test setup

In the example the test process constitutes as follows:
The user sends a test request to the TraceQoS manager (TM) at the source (TS0), which will check if it willing to participate as a sender (test request to itself). Then it sends a receiver setup request to TS1, which also evaluates if it is willing to participate as a receiver. It will confirm and set up a test server. After receiving the confirmation, the source (TS0) then initiates the test sender which sends test data to TS1. It then sends back a confirmation to the TM with information of where to find the measurement results.

Then the TM starts a test request to TS1 to initiate a test session with TS2. The proceeding is exactly the same as above, just with machines (TS1, TS2) instead of (TS0, TS1).
The same is done for test between (TS2, TS3), TM sends a test request to TS2 to initiate a test session with TS3.

After all tests have been set up, the TM starts to gather the test data from the participating TraceQoS servers, analyses the data and delivers results to the user.

Please note that tests are all being carried out in parallel, as the next test is always initiated as soon as the previous test is confirmed.
Also test data can be fetched while the tests are still in progress. All data is streamed to the TM until there is no more data.

6.2 Router Discovery

To find the routers on the path from the source host to the destination, the tool Traceroute [28] is used. To allow discovery of a path over IPv6 as well as IPv4, a Traceroute implementation that supports both IPv4 and IPv6 is needed. The Traceroute discovery process does not always lead to all the routers on the path, as some routers do not send an ICMP time exceeded message back to the originating hosts, when the TTL field expired. Consequently, it may be that TraceQoS does not have a complete list of routers on the way from source to destination. Unfortunately there is no other known method to gather a complete list.

6.3 TraceQoS Server Discovery

For each router that one would like to include in a measurement, a corresponding server as close as possible has to be set up. Keep in mind that each TraceQoS server can only be associated to exactly one router[22] (see 5.3).
For a mapping of routers to their associated TraceQoS servers, mapping information (e.g. router IP, server IP, server port) has to be stored somewhere. As a router usually has more than one interface, each interface should have its own entry. For TraceQoS servers it is sufficient to provide the associated hostname[23].
A TraceQoS server directory could be done in several different manners; some ideas are:

6.3.1 Manually Edited List

A list of all routers with their associated TraceQoS server is kept in a local file and edited manually[24]. This is what the first prototype implementation uses.

6.3.2 Centrally Maintained List

A list of all routers with their associated TraceQoS server is fetched at the start up of the TraceQoS manager from a central location.

6.3.3 Domain Name System (DNS)

If integrated within DNS directory services [33], with the use of a new resource record type, associating TraceQoS servers with a router should be an easy thing to do. A simple name lookup[25] will be sufficient to get an association. However this requires the use of one or more DNS servers.

6.3.4 Peer-to-Peer Discovery

Using some sort of distributed service discovery. This might involve broadcast and/or multicast messages.
For example, if you have a set of known TraceQoS servers, you send a query for a specific router to all of them, which then forward it to the associated TraceQoS server (if they have that information) or send it to all other TraceQoS server they know. Duplicate queries are discarded. Once the responsible TraceQoS server for that query gets a request, it answers directly to the inquiring host. (Another variant would be backtracking with route recording, so that all other TraceQoS servers on the way could update their database)
To guarantee consistency, answers may only be sent by the TraceQoS server that is associated with the given router interface.

6.4 Measurement Procedure

Actual measurement on a single link is being carried out between two TraceQoS server, one running a test sender and the other a test receiver.

[22] All interfaces of that router have to be specified.
[23] All TraceQoS servers are required to have a DNS hostname assigned.
[24] Similar to hosts.txt
[25] A query to a DNS server to request a certain resource record

A standard TraceQoS test measures one-way delay, one-way delay variation and packet loss of IP traffic between two TraceQoS measurement stations. To allow other sorts of tests to be implemented in the future, the TraceQoS protocol features a test type field in the test set-up request.
The TraceQoS test sender sends a negotiated number n of packets of a negotiated fixed size s to the test receiver (a test with variable-sized packets with lower and upper bounds could be considered for future work).
The rate at which packets are sent is also indicated at the test set-up request.
The test packets sent need to include the following information: (excluding IP and UDP header)

Test ID,
Sequence Number,
Source Timestamp (in microsecond),
Padding Data (to accommodate for a given size of a test packet),

A sequence number is kept to keep track of packets that are lost.
The source timestamp is for calculating delays.
Padding data is needed to gain the specified length of packet. It should contain random bits to avoid compression along the path.
Whether a checksum is needed has to be further investigated. It is currently assumed that test packets will already be encapsulated in UDP packets with activated CRC32 checksum.

The test receiver receives the packets sent by the test sender and stores information of each packet along with a timestamp of receipt locally.

The packet log is to contain all "header" and receipt timestamps of all packets, along with a session ID that identifies the test.

Example for a packet log content:

Packet log header:

Test ID
Number of packets

Following are packet information for each packet received:

Sequence Number,
Packet Size,
Source Timestamp,
Destination Timestamp.

6.5 Test Results

Test Data is fetched from participating TraceQoS servers by the TM.
The TM calculates the needed metrics and provides the client with a dynamic view of the currently received test data and a summary thereafter.

For example a summary could contain the following information:
Min. one-way delay in ms
Max. one-way delay in ms

Mean one-way delay in ms
Min. one-way delay variation in ms
Max one-way delay variation in ms
Mean one-way delay variation in ms
Packet loss ratio (packets lost / packets received)

What information is considered valuable is a detail of the implementation.

To allow a more intuitive interpretation of the test results, a graphical view of the calculated metrics over the test duration is desirable.

7 Security Considerations

A goal of the TraceQoS infrastructure is to provide anonymous access to measurement stations. Consequently, possible security risks that arise by allowing anonymous use have to be considered and eliminated if possible.
The following risks have to be evaluated:

- Denial of Service attacks
- Distributed Denial of Service attacks

7.1 Denial of Service (DoS) attack

First of all, a TraceQoS server machine should solely be used for TraceQoS and not for other relevant purposes, so a denial of service attack can not compromise the functionality of other vital applications.
Furthermore, the following actions have to be taken to reduce the risk of a denial of service attack on a TraceQoS server:

- The number of test requests that are accepted from a single host within a given time frame should be limited to a reasonable threshold.
- The packet rate of a single test has to be limited to a reasonable threshold, as otherwise one test could consume all the resources of a TraceQoS server, leaving no room for other concurrent tests.
- System resources needed for a particular test have to be freed upon completion or abortion of a test. This does not include test results on disk, for which a separate rule exists.
- Test results need to be deleted regularly to avoid exhaustion of disk space. However, enough time should be allowed after the completion of a test to fetch the results by the caller.

Additionally, before accepting a test, the following criteria are checked by a TraceQoS server,

- IP of sender (blacklist)
- number of current sender processes
- number of current receiver processes
- current total sender rate
- current total receiver rate
- system resources

7.2 Distributed Denial of Service (DDoS) attack

To avoid contributing to a distributed denial of service attack platform, each test has to be confirmed by the sender as well as the receiver of the platform. In this way it is impossible to send test traffic to some host that is not willing to participate. If a TraceQoS server does not respond to a start test request, no retry attempts are made. This way a TraceQoS server can not flood a remote TraceQoS server with the same requests. As the number of active test requests per host is limited according to 7.1, sending multiple requests in short time intervals do not add to the danger as they are simply declined at the receiving host.

8 TraceQoS protocols

The TraceQoS process is divided in 5 sub-processes:

- The TraceQoS Manager (TM), which organises the whole TraceQoS process
- The TraceQoS Sender Control (TSC), which responds to TraceQoS test requests and sets up the test sender (TTS)
- The TraceQoS Receiver Control (TRC), which responds to receiver set-up requests and sets up the test receiver (TTR).
- The TraceQoS Test Receiver (TTR), receives test packets and stores results locally.
- The TraceQoS Test Sender (TTS), which sends test packets to the TraceQoS Test Server.

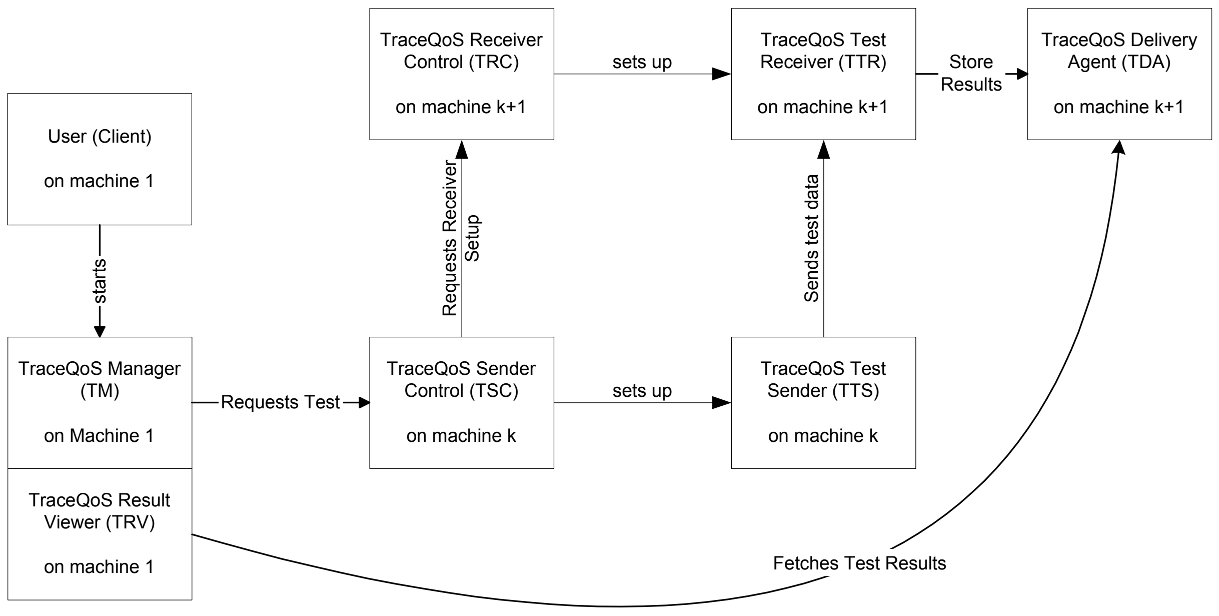


Figure 9: Protocol Overview

8.1 Protocol Overview

8.1.1 TraceQoS Manager (TM)

1.) Receives request for a TraceQoS from the machine it is running on to a specified destination.
2.) Check if test is acceptable.
3.) Discover all routers on the path from Src to Dst. (R1, R2, ... Rn)
4.) Discover associated TraceQoS servers. (TS1,TS2,..., TSm) (m<=n)

For each of the discovered servers do:

5.) Send test request to TSx (for measurement from TSx to TSx+1).
6.) Wait for confirmation from TSx.

Then after all tests have been confirmed, do:

7.) Start the TraceQoS Result Viewer (TRV) to fetch results

8.1.2 TraceQoS Sender Control (TSC)

1.) Wait for incoming test request
2.) Decide if willing to participate in test. (Resource Control)
3.) Send test request to TRC on destination
4.) After confirmation from TRC, confirm test to caller.
5.) Call TTS on local machine to perform test

8.1.3 TraceQoS Receiver Control (TRC)

1.) Wait for incoming test request from a TSC
2.) Decide if willing to participate in test.
3.) If acceptable, call TTR on the local machine.
The machine is then ready to receive test traffic.
4.) Send test confirmation to the calling TSC

8.1.4 TraceQoS Test Receiver (TTR)

1.) Wait for incoming test packets
2.) Timestamp test packets at arrival
3.) Store packet trace in a designated temporary file.
4.) Close socket, exit.

8.1.5 TraceQoS Test Sender (TTS)

1.) Send test packets to the TTR.
2.) Exit

The format of the test packets depend on the test to be performed.

8.1.6 TraceQoS Result Viewer (TRV)

1.) Get test data from the participating TraceQoS servers.
To identify the test to be retrieved, the unique Test ID is used.

2.) Analyse the data and calculate results
3.) Present the results to the user

8.2 Establishing a TraceQoS test connection (TM -> TS1)

The TraceQoS control protocol is vaguely based on the OWDP Implementation by Kalidindi [4].

The part of TraceQoS used to establish a test-connection is essentially a simple request-reply exchange. Although only little information is transmitted, a TCP connection is used as a transport layer for the connection setup. This ensures that no duplicate or lost requests will exist, and thus simplifies the protocol.

The following is a narrative description of the protocol in the "normal case".

TraceQoS manager initiates the protocol by requesting a test-connection. It sends a "Start Test Request" (STRQ) message to TraceQoS Send Control (TSC).

The following information is provided by TraceQoS Manager (TM) to TraceQoS Send Control (TSC) in the STRQ message:

Version	The version number of the TraceQoS protocol
Test-ID	Identification of the current TraceQoS session.
Destination-Host	Host name of the TRC. (FQDN)
Destination-Port	Port of the TRC.
Address-Family	IP Protocol Version to use for test traffic
Test-Port	Desired port of the test traffic receiver.
Test-Type	Identifies the type of test packets used for test traffic. Standard is fixed size UDP packets.
Test-Packet-Size	Size of the test packets to send in bytes. The minimum value depends on the implementation and on what information is included in a test packet. The maximum value is limited by the maximum size of an IP packet.
Test-Packet-Rate	How many packets per second are sent by the TraceQoS test sender (TTS). The minimum value is 1 packet per second.

When TSC receives an STRQ message, it first checks the request for validity. The following checks are made:

- If the Version does not match, a corresponding error message is returned and the connection is closed. (e.g. VERSION_UNSUPPORTED)

- Admission Control: TSC must determine that it is willing to engage in a TraceQoS test to the TraceQoS test destination. This can include a check of available resources, identity of the calling entity, current active connections, etc. and is a detail of the implementation and configuration. If the request does not pass this check, an appropriate error message has to be returned and the connection is closed.

If the STRQ message is valid, the following actions are taken:

TSC sends a Start Receiver (SRCV) message to the destination host to see if the TRC (TraceQoS Receiver Control) is willing and ready to participate in the test.

The SRCV message contains the following information, which is mainly the same as the he original STRQ message that TSC received:

Version	The version number of the TraceQoS protocol
Test-ID	Identification of the current TraceQoS session
Test-Port	Desired port of the test traffic receiver.
Test-Type	Number that identifies the type of test packets used for test traffic. Standard is fixed size UDP packets.
Test-Packet-Size	Size of the test packets that are sent in bytes. The minimum value of STRQ-Test-PacketSize depends on the implementation and on what information is included in a test packet. The maximum value is 65535 octets. (The maximum length of an IPv4 packet). For IPv6 the maximum value is 65735.
Test-Packet-Rate	How many packets per second are sent by the TraceQoS test sender (TTS). The minimum value is 1 packet per second.

When TRC receives an SRCV message, it first checks the request for validity. The following checks are made:

- If the Version does not match, a corresponding error message is returned to the calling TSC and the connection is closed. (e.g. VERSION_UNSUPPORTED)

- Admission Control: TRC must determine that it is willing to engage in a TraceQoS test to receive traffic from the calling entity. This can include a check of available resources, identity of the calling entity, current active connections, etc. and is a detail of the implementation and configuration. If the request does not pass this check, an appropriate error message (e.g. TOO_MANY_CONNECTIONS) has to be returned to the calling TSC and the connection is closed.

If the SRCV message is valid, the following actions are taken:

TRC creates a TraceQoS test receiver (TTR). TTR is initialised, using information from the SRCV message. A UDP port, termed TTR-Port is allocated to TTR.

At this point TTR is prepared to receive the sequence of test messages. TRC now sends an "ok to receive" (OKTR) reply to TSC.

The OKTR message includes the TTR-Port to which the TTS should send the test packets. (It could differ from the requested destination Port if it is already in use by another test session)

When TSC receives the OKTR message, it initializes the TTS using the TTR-port.
At this point, the test-connection is established and test packets are being sent.
TSC now sends an "ok test started" (OKTS) message to TM.

When TM receives the OKTS message, it continues to request the remaining tests.
In case TM receives an error message, it might use the error information to request another test from the same host with different parameters. Usually this is not wanted by the user though, and the TM tries to initiate a test with the next server in line.

8.3 TraceQoS Test Traffic

Once a test-connection is set up, TTS sends test packets (TEST) to TTR, following a Poisson schedule with an average rate lambda=1/packetRate[26].

In the TEST packets, the TTR-port is used as the UDP Destination port and TTS-port as the UDP Source Port.

The following information is included in the TEST packets.
- The Test-ID
- A sequence number, TTC-Sequence-Number. Sequence numbers start from 1.
- A timestamp, TTC-Timestamp. (includes date and time, in microseconds)
- An error estimate (in microseconds)
- Padding data

As soon as TTR receives a test packet, it takes a timestamp, referred to as TTR-Timestamp.
The following checks are made by TTR:
- It checks that the source IP in the IP header and the UDP source port correspond to the test-connection. It is discarded if the packet fails the check.
- It checks if the Test-ID matches the test connection. It is discarded if the packet fails the check. If the packet passes the check, TTR stores the packet data locally where it can be extracted by the TraceQoS Delivery Agent (TDA) by the Test-ID.

[26] The test packets are sent following a Poisson-distribution to reflect a „random" traffic pattern, which should help to recognize periodic network failures. If the test packets were sent in fixed intervals, a periodic anomaly could remain undetected. See [36] for more details on this subject.

9 Prototype Implementation

For the prototype we decided to use HTTP as transport vehicle for the control protocol, which is implemented as CGI scripts that are called via a public web server on TraceQoS servers. This simplifies setup of TraceQoS servers, and allows the protocol data to pass most firewalls as it is considered as "web traffic".

The code for actual measurements and storage of result is written in C, using socket libraries, compiled with GCC. Sample code from [1] and [37] was used as groundwork. Furthermore the tool qosplot [19] is used for graphical output.

The CGI control scripts call the compiled C code for sending and receiving of test data.

9.1 Overview

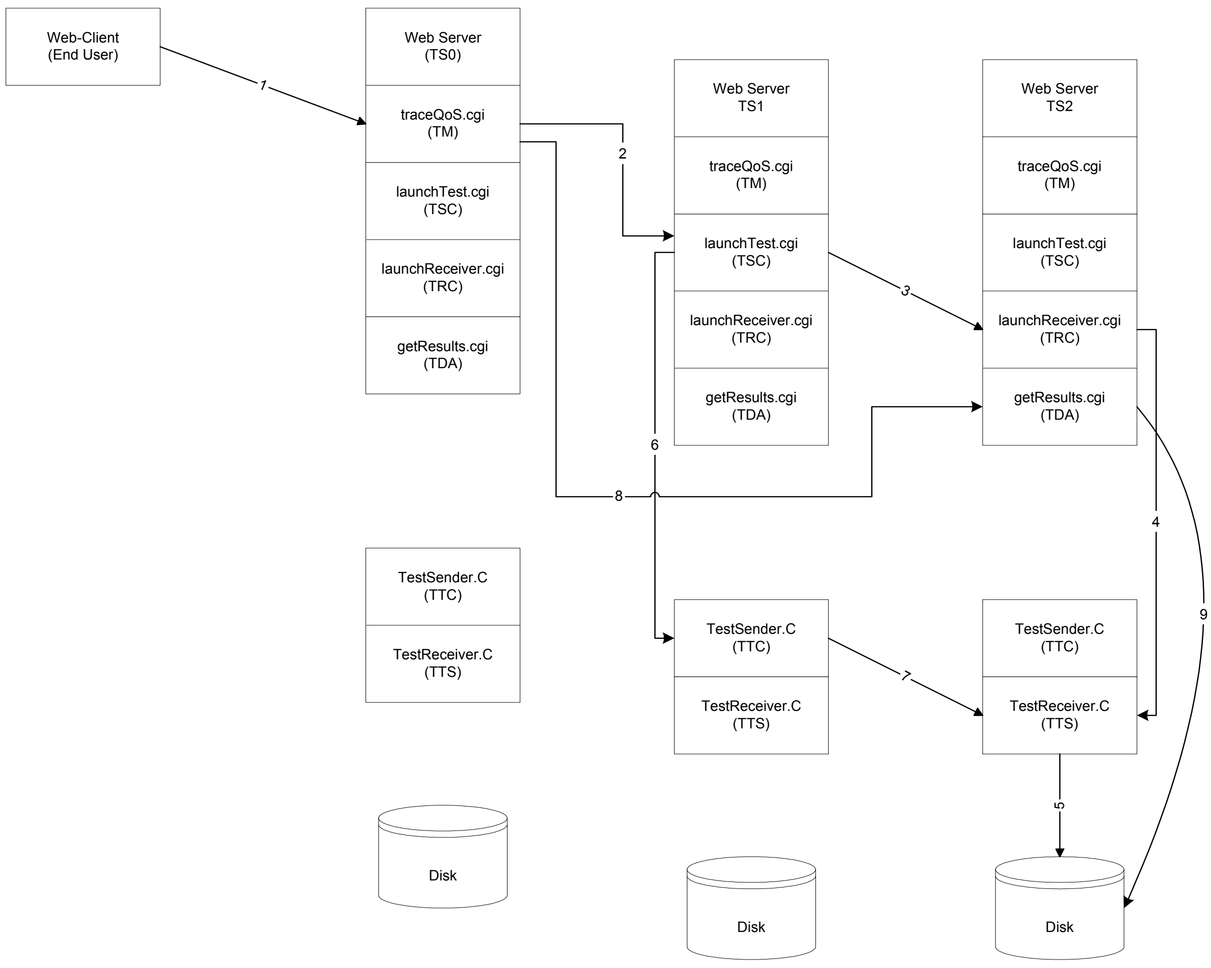


Figure 10: Flow diagram for a test session between TS1 and TS2

9.2 Components

9.2.1 Traceqos.cgi

Description:
Client application that coordinates a TraceQoS test session.

Input Parameters:
- Destination host name or literal address (IPv4/IPv6)
- Preferred address family (IPv4/IPv6)[27]
- Test port
- Test type
- Number of packets to send
- Packet size
- Packet rate

Return Parameters:
- Dynamically display results of incoming data
- Display a graphical summary at the end in a web browser.

Operating Sequence:
1. Find routers from TM host to destination host
2. Find associated TraceQoS servers
3. For each TraceQoS server found
   Request test -> call LaunchTest.cgi
4. View Results:
   a) For each confirmed test between two single TraceQoS servers, get results
   b) Analyse data
   c) Output results

9.2.2 LaunchTest.cgi

Description:
Initiates a single test between two TraceQoS servers. This is hosted on the sending server.

Input Parameters:
- Test-ID
- Receiver host name or literal address (IPv4/IPv6)
- Preferred address family (IPv4/IPv6)[27]
- Receiver port
- Test type
- Number of packets to send
- Packet size
- Packet rate

Return Parameters:
- Confirmation/Denial

Operating Sequence:
1. Check if request is valid
2. Call LaunchReceiver.cgi to see if the receiver is willing to participate in test.
3. Start TestSender module to send packets to the receiver.
4. Confirm test to caller

9.2.3 LaunchReceiver.cgi

Description:

[27] Required if no literal address is given.

Used to start the receiving end of a single TraceQoS test connection.

Input Parameters:

- Test-ID
- Receiver port
- Test type
- Number of packets sent
- Packet size
- Packet rate

Return Parameters:

- Confirmation/Denial

Operating Sequence:

1. Check if request is valid
2. Call TestReceiver.C (TTR)
3. Confirm readiness of test receiver to caller

9.2.4 GetResults.cgi

Description:
Reads the results from the local DB file and returns them dynamically as new data gets available.

Input Parameters:

- Test-ID

Return Parameters:

- Test data (The format of the streamed result data has yet to be determined)

Operating Sequence:

1. Read data from local DB
2. Return it to caller

9.2.5 StopTest.cgi

Description:
Stops sending and receiving processes according to Test-ID supplied.
NOTE: This feature has not been implemented yet!

Input Parameters:

- Test-ID

Return Parameters:

- Confirmation

Operating Sequence:

1. Look up Process ID's of the processes associated to the Test-ID
2. Send kill signal to processes
3. Confirm abort to caller

9.2.6 TestReceiver.C

Description:
Listens to packets and stores data about them in a local DB. After a given timeout the program assumes no more packets will arrive and exits. There is no other solution than a timeout because you can never be sure when the last packet arrived (as there can be lost packets, re-ordered packets and duplicates).

Input Parameters:
- Receiver port
- Test type
- Packet size
- Packet rate
- Timeout value
- Test ID

Return Parameters:
- None

Operating Sequence:
1. For each incoming test packet until idle time out, do:
   a) get timestamp when packet arrives
   b) save packet data information to local DB
2. After the idle timeout expired, mark the end of the test in DB and exit

9.2.7 TestSender.C

Description:
Sends packets according to the Input Parameters.

Input Parameters:
- Receiver host name or literal address (IPv4/IPv6)
- Preferred address family (IPv4/IPv6)[27]
- Receiver port
- Test type
- Number of packets to send
- Packet size
- Packet rate

Return Parameters:
- None

Operating Sequence:
1. Lookup host name according to address family
2. For each packet to send:
   a) Form test packet
   b) Take current timestamp and add it to the packet
   c) Immediately send packet to the receiver host

9.2.8 TraceQoS Server Directory

In the prototype implementation, the server directory is very simple. A text file called "serverlist" is used to associate routers with TraceQoS servers. It has to be in the same directory as the traceqos.cgi.

```
# traceqos server list
# router name / traceqos server / traceqos server port / cgi dir

santorini.switch.ch     ezmp2.switch.ch  80  /cgi-bin/traceqos/
swiCS3-V4.switch.ch     ezmp1.switch.ch  80  /cgi-bin/traceqos/
w3.ethz.ch              lsmp1.switch.ch  80  /cgi-bin/traceqos/
...
...
```

All interfaces of a router have to be included on a separate line each.

9.2.9 TraceQoS Result Viewer

Results are cumulatively displayed dynamically as new test data arrives. When all test data has been collected or a timeout requesting test data from servers expired, a visual representation for each test connection is generated and can be viewed by a web-browser.

9.3 Current Restrictions

- A list of routers / TraceQoS servers associations has to be provided in a text file.
- Test traffic is restricted to fixed size UDP packets.
- Concurrent tests have to use different port numbers.
- Due to a buffering/flushing problem on apache 2.0 web servers, the minimum packet rate for tests should be around 50 packets per second to avoid time outs at the TM. This only applies to apache 2.0 web servers.
- Lacking error handling
- No resource control

9.4 System Requirements

9.4.1 Client

- OS : Unix Sun Solaris 5.8 or Linux 2.4
- Perl 5.6
- Perl Libraries :
  CGI, Time ::HiRes, HTTP ::Request ::Common, LWP ::UserAgent, LWP::Parallel::UserAgent
- Traceroute, executable by current user id
- Netscape Browser 4.7 or higher

9.4.2 Server

- OS: Unix Sun Solaris 5.8 or Linux 2.4 (these operating systems were tested, other similar systems might also work)
- Perl 5.6
- Perl Libraries:
  CGI, Time ::HiRes, HTTP ::Request ::Common, LWP ::UserAgent, File::Tail
- GNU C compiler to build test sender and test receiver module.

- Web Server that supports CGI-BIN (and non-parsed headers). Apache 1.3 is recommended for this purpose.
- Directory writable by CGI to hold temporary test data.

9.5 Installation

Before you try to install TraceQoS, please remember that this software is a prototype in alpha version. It has not been tested in various configurations, and there are still lots of bugs present. This is a work in progress, and will be developed further on.

9.5.1 Server

- Create a subdirectory for TraceQoS in the CGI-BIN of your webserver. (e.g. /cgi-bin/traceqos)
- Copy all .cgi files to that directory
- Compile testsender.c and testreceiver.c (make all)
- Copy testsender and testreceiver executables to the TraceQoS CGI directory
- Make all those files readable for anonymous web server user (www-user)
- Create a directory for temporary data files, writeable by CGI-scripts (usually this is for www-data) (e.g. /var/local/traceqos)
- Edit configuration file to reflect your configuration

9.5.2 Client

- Make sure a current version of gnuplot with png support, and perl 5.6.0 or higher is installed. Also Netscape is needed for graphical summary.
- Create a subdirectory for TraceQoS (further referred to as TraceQoS directory)
- Copy all client files into the TraceQoS directory (traceqos.pl, gnuplot.commands, gnuplot.commands2, gnuplot.data, index.html, serverlist)
- Edit serverlist to add new known TraceQoS servers
- Compile the latest version of qosplos.C for your OS and copy the executable into the TraceQoS directory

9.6 User Interface

The application is started by calling the perl script "traceqos.pl". It is currently a CGI script, so the parameters have to be in param=value format.

Current Parameters:

destinationHost:	hostname of the destination
nrOfPackets:	number of packets to send in one test
packetSize:	size of the test packets[28]
packetRate:	number of packets per second to send[29]
addressFamily:	IP version to use for test traffic (0=unspecified, 1=IPv4, 2=IPv6)

Example of a command line :

```
Perl traceqos.pl destinationHost=www.unige.ch nrOfPackets=200 packetSize=140 packetRate=10
addressFamily=1
```

[28] This value indicates only the payload size
[29] The actual rate will be only an approximation of this value

This starts a test session between all the known TraceQoS servers on the path from the current host to www.unige.ch. Between adjacent TraceQoS servers 200 packets with a packet size of 140 bytes are sent with a approximated packet rate of about 10 packets/second.

Note: Currently there is a bug in the buffering of the result data at the web server[30]. The data is only returned in large chunks of about 8Kbyte. This way a high packet rate is needed to prevent a time-out of the fetching of test data by the client. This problem is being investigated.

Example screen of a test setup:

```
Welcome to TraceQoS prototype alpha 0.01

Protocol Version: 1
Destination Host: www.unige.ch
Test Port: 34671
Test ID: 1042472251.603692
Test Type To Perform: 1
Number Of Packets To Send: 200
Packet Size: 140 bytes
Packet Rate: 10 packets/s

Finding routers on the way to www.unige.ch...
traceroute to silene.unige.ch (129.194.8.27), 30 hops max, 40 byte packets
swiCS3-V4.switch.ch
swiEZ2-G3-3.switch.ch
swiCE2-G2-2.switch.ch
swiCE1-G0-0-0.switch.ch
swiGE1-A0-0-0-2.switch.ch
xtrn-eth1.unige.ch
129.194.144.2
silene.unige.ch

Looking for associated TraceQoS servers...
found ezmp1.switch.ch:80 /cgi-bin/traceqos/ for swiCS3-V4.switch.ch
found ezmp2.switch.ch:80 /cgi-bin/traceqos/ for swiEZ2-G3-3.switch.ch
no associated traceQoS server found for swiCE2-G2-2.switch.ch
no associated traceQoS server found for swiCE1-G0-0-0.switch.ch
found lsmp1.switch.ch:80 /cgi-bin/traceqos/ for swiGE1-A0-0-0-2.switch.ch
no associated traceQoS server found for xtrn-eth1.unige.ch
no associated traceQoS server found for 129.194.144.2
no associated traceQoS server found for silene.unige.ch

Path Information:
Router / Associated TS / Port Number
------------------------------------------------------------
swiCS3-V4.switch.ch ezmp1.switch.ch
swiEZ2-G3-3.switch.ch ezmp2.switch.ch
swiGE1-A0-0-0-2.switch.ch lsmp1.switch.ch

Setting up tests between TraceQoS servers...
Sending request to: http://ezmp1.switch.ch:80/cgi-bin/traceqos/nph-launchTest.pl
Test Request accepted.
Sending request to: http://ezmp2.switch.ch:80/cgi-bin/traceqos/nph-launchTest.pl
Test Request accepted.
No more servers to request tests
```

Results of the test connections between the TraceQoS servers are displayed dynamically as tests are running (with the exception noted above).
Example of dynamic output while test is in progress:

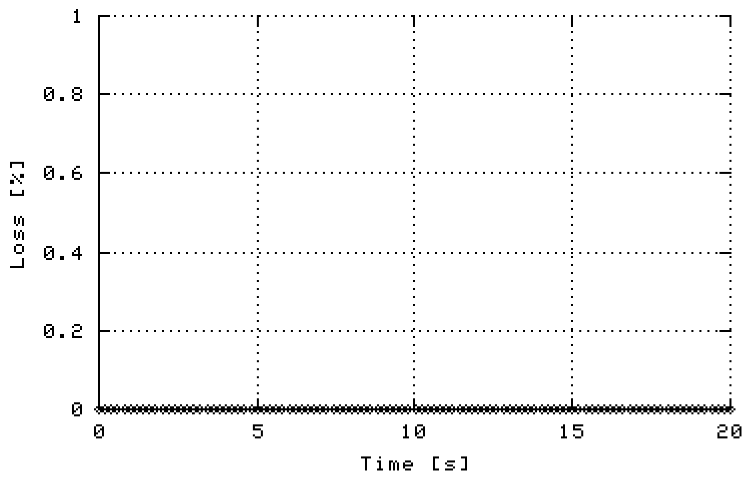
```
TRACEQOS V0.01 ALPHA

Test ID: 1042472251.603692 Test Type: 1 (UDP, fixed size)
Packet Size: 140 bytes, Packet Rate: 10 (packets/sec), IPv4
```

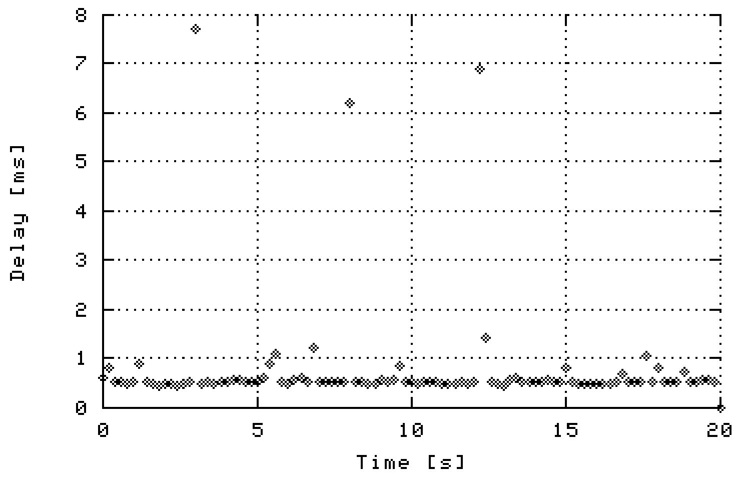
[30] This phenomena occurred using an Apache 2.0 web server on an Intel Linux machine. It appears that Apache 1.3 does not have this limitation, so using Apache 1.3 or an equivalent web server is recommended.

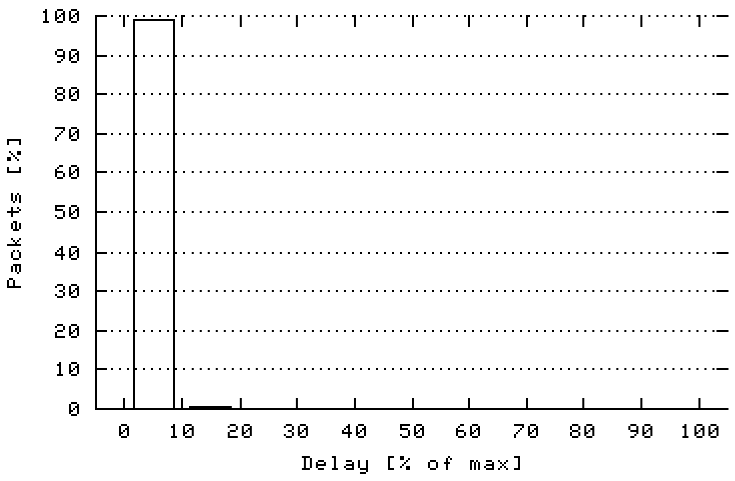
```
ezmp1.switch.ch (swiCS3-V4.switch.ch)
 to ezmp2.switch.ch (swiEZ2-G3-3.switch.ch):
Packets(total/received/lost/loss ratio): 200 / 200 / 0 / 0.00
OWD ms  (min/avg/max): -0.734 / -0.716 / -0.597
OWDV ms (min/avg/max): 0.035 / 0.083 / 0.101

ezmp2.switch.ch (swiEZ2-G3-3.switch.ch)
 to lsmp1.switch.ch (swiGE1-A0-0-0-2.switch.ch):
Packets(total/received/lost/loss ratio): 200 / 200 / 0 / 0.00
OWD ms  (min/avg/max): 1.220 / 1.233 / 1.305
OWDV ms (min/avg/max): 0.002 / 0.031 / 0.044
```

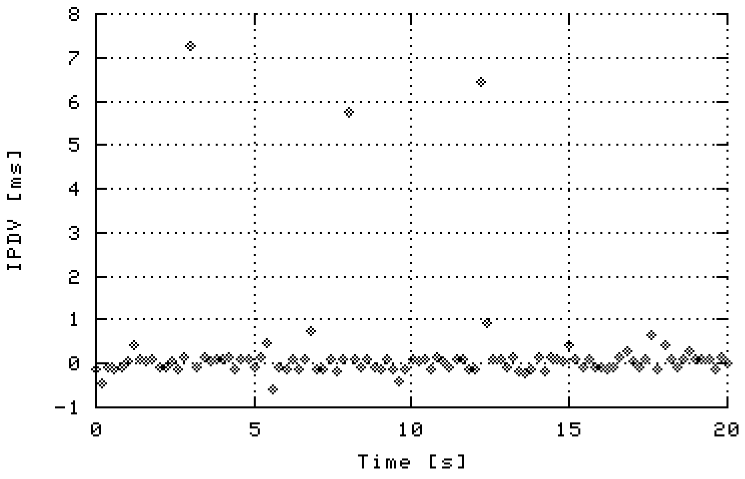
After all tests have completed, a visual representation of the test traffic is generated and displayed using a Netscape Internet browser.

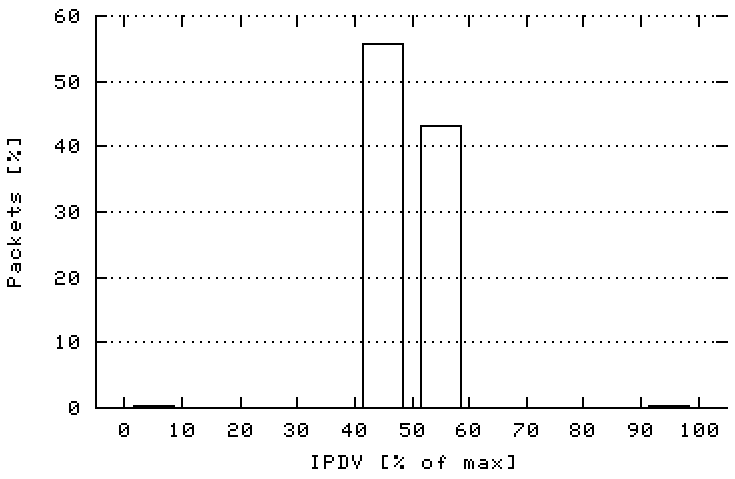
Example of graphical output after tests completed: (This only covers the test connection between ezmp2.switch.ch and lsmp1.switch.ch, as it would take to much space in here to show both)

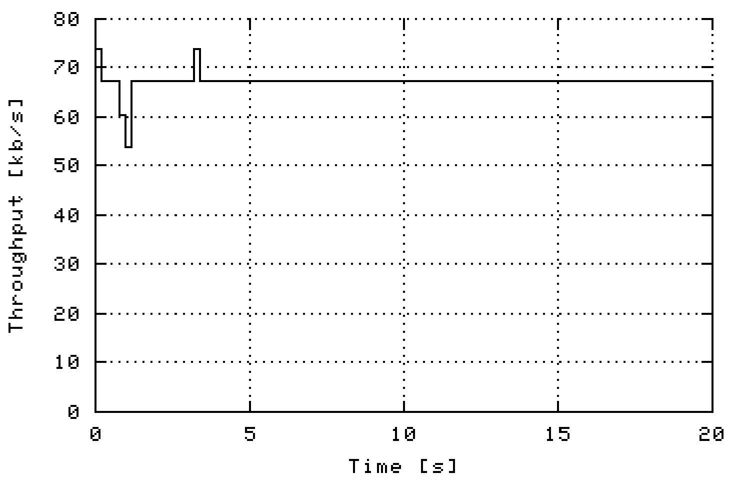





Packets [%]
Delay [% of max]
0 10 20 30 40 50 60 70 80 90 100


IPDV [ms]
Time [s]
0 5 10 15 20


Packets [%]
IPDV [% of max]
0 10 20 30 40 50 60 70 80 90 100



9.7 Interesting Test Results

9.7.1 Time Drifting

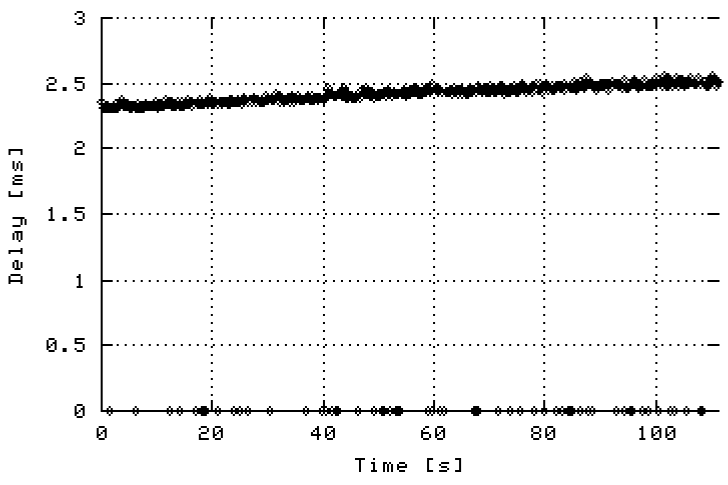


Figure 11: Time drifting with NTP

In Figure 11, we can see an almost linear increase in delay during an interval period of 110 seconds. 1000 packets were sent with a packet rate of approximately 10 packets a second. The observed connection can be considered stable, with delays that should remain constant. Thus, it can be assumed that the steadily increasing delay shown in the graph is a result of the clocks drifting away from each other on the two participating hosts.

9.7.2 Busy router

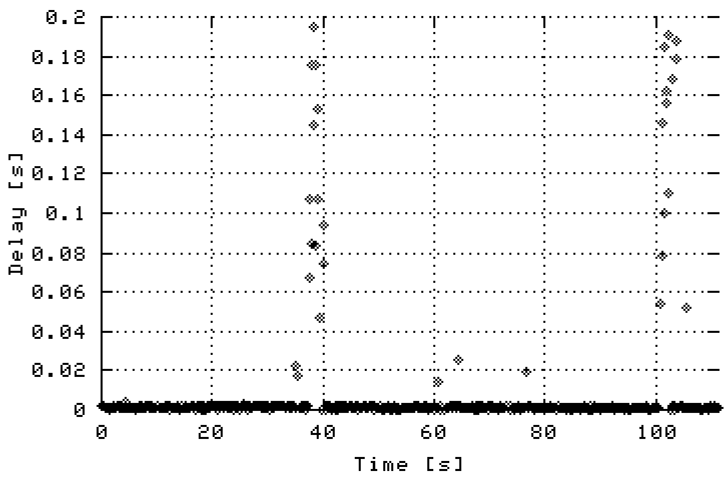


Figure 12: IPv6 router busy with BGP updates

The delay pattern seen in Figure 12 with sudden huge delay increases was observed when sending IPv6 test traffic over a IPv6 router without fast-path enabled. As the anomalies occurred steadily about every 60-64 seconds, it is assumed that this may be the result of the router CPU being busy with BGP routing table updates, thus delaying packet forwarding.

10 Conclusion

A protocol for the purpose of TraceQoS has been developed and implemented as a proof of concept in form of a prototype. With the prototype it is possible to launch distributed measurements between adjacent TraceQoS servers and to poll the results at one location.
The prototype was deployed on 3 test machines located in the network of SWITCH, running under Linux and Apache 1.3 as a web server.
Current measurements have shown that NTP synchronisation is limited to an accuracy of around 1-5 ms even on a LAN, as previously expected. Thus, calculated one-way delays are often quite inaccurate, especially if distances (and transmission delay) are short. If one-way delay accuracy is of more interest, the use of GPS devices for time synchronisation is recommended. However, to discover QoS failures, the accuracy of the measurements seem to be sufficient.
Thanks to the visual representation of the measurement data, some interesting phenomena like time drifting and busy routers during BGP updates could be observed.
The TraceQoS tool has received positive feedback so far and future work is planned in the following areas:

- Packaging and further deployment
- Improvement of error handling
- Allowing concurrent tests on the same port (de-multiplexer)
- Security/Access/Resource/Rate control with feedback mechanism to prevent (D)DoS attacks
- Rate control based on network feedback (e.g. packet loss as congestion indication)

Appendix A: Abbreviations

ACK	Acknowledgement
ATM	Asynchronous Transfer Mode
BSD	Berkely Software/Standard Distribution
CDMA	Carrier Detection Multiple Access
CGI	Common Gateway Interface
CPU	Central Processing Unit
DB	Database
DDoS	Distributed Denial of Service
DNS	Domain Name System
DoS	Denial of Service
Dst	Destination
FQDN	Fully Qualified Domain Names
FTP	File Transfer Protocol
GCC	GNU C Compiler
GNU	Gnu's Not Unix
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
ID	Identification
IDS	Intrusion Detection System
IP	Internet Protocol
IPMP	IP Measurement Protocol
IPv4	IP Version 4
IPv6	IP Version 6
NIC	Network Interface Card
NTP	Network Time Protocol
OS	Operating System
OWD	One Way Delay
OWDV	One Way Delay Variation
OWPL	One Way Packet Loss
QoS	Quality of Service
R	Router
RCS	Radio Control Signal
RFC	Request For Comments
RTT	Round Trip Time
Src	Source
TM	TraceQoS Manager
TRC	TraceQoS Receiver Control
TRES	Test Results packet
TRV	TraceQoS Result Viewer
TS	TraceQoS Server
TSC	TraceQoS Sender Control
TTL	Time To Live
TTR	TraceQoS Test Receiver
TTS	TraceQoS Test Sender
UDP	User Datagram Protocol
UTC	Coordinated Universal Time
WAN	Wide Area Network

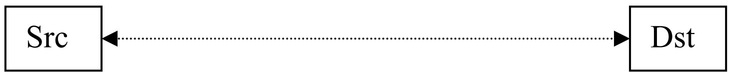
Appendix B: One-Way Delay Measurement and Time Synchronization

Extract from [18]: “The one-way delay metric, as defined in [26], is one of the basic quantitative characteristics of the network propagation delay. It is preferred to round-trip delay as there may be a significant asymmetry in paths to the destination and back to the source and even if the two paths are symmetric, they may have different performances due to queuing. The metric is defined as the difference between wire-time of the first bit of a packet at the transmitter and wire-time of the last bit at the receiver. The metric involves an upper bound of delay and says that packet is considered lost and the value of metric is undefined if the last bit does not arrive within that predefined period of time. If the packet is fragmented and if, for whatever reason, reassembly does not occur, the packet will be deemed lost. Note that measuring one-way delay requires clock synchronization between the sender and receiver.”

IP traffic nowadays is highly asymmetric, which is the reason why network providers/operators prefer to have one-way performance measurements.
Unfortunately, there is a big problem associated with one-way measurements. All hosts involved must have synchronized clocks.
Usually this is achieved by synchronising the hosts with NTP servers that have the exact time from an external time source (RCS (radio control signal)/GPS/CDMA). Unfortunately this is not highly accurate as the delay to the NTP servers itself is not constant and so currently only synchronization in the range of milliseconds (ms) can be achieved. In lightly loaded LAN's and short-distance WAN, synchronization with a stratum-1 server can result in synchronization under a millisecond [16]. For more precise measurement in the microseconds (µs) range, directly attached GPS devices may be used at the test hosts.

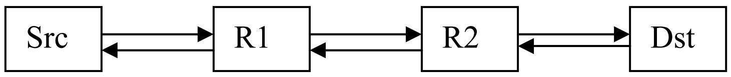
Appendix C: Model of Network environment

C.1 End-to-End view



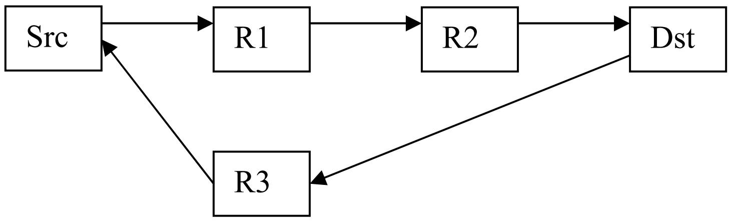
This is the typical view of an end-user. It explicitly simplifies the view to the source and the destination. Routers on the way to the destination are not visible. Performance measurements take place only directly between Src and Dst. The end-user is only interested in round-trip time, in other words, “how long does it take until I receive an answer from Dst?”. On the IP level this is usually measured by using the tool “ping” [32].

C.2 Link-to-Link View



This is an example of a "real world" model of an IP path. An IP packet passes Router R1 and R2 on the way to the Destination Dst.
Note: Although in this example the path is the same for both directions, this has not to be the case at all.

For example it might look like this:



In this case the return path is completely different, which also could result in completely different delay, jitter and loss characteristics.
Thus tools that are based on two-way performance measurements could easily deliver deceiving results. (E.g. ping, Traceroute).

Appendix D: Time-Stamping the test packets by user-level processes

(This is chapter 4 of [35])

As stated in the Metric definition drafts for measuring one-way delay and packet loss, test-packets are time stamped at both the source and the destination. The IPPM Framework document and the metric drafts describe how the accuracy of the measurements need to take into account *wire-time.* Essentially, the measurement techniques need to avoid introducing artificial performance problems caused due to load on the measurement machine itself. So test packets need to be time stamped as close to the wire as possible, at both the source and the destination. If the timestamps are different from the wire-time, the measurements would have errors and a calibration of the measurement architecture would yield large error bars.

The current measurement software is user-level C code written using 4.4BSD socket layer libraries. [Note: The TraceQoS prototype implementation uses Sun OS 5.8 socket libraries resp. Linux sockets]. The measurement agents are user-level processes. User-level software is simple to implement, but as we shall describe below, user-level measurements introduce errors in measurements.

After the time stamping at user-level, the packet still needs to traverse the following path through the kernel before it reaches the wire:

The user-level process needs to hand the packet to the kernel transport-layer protocol handler.
The transport-layer protocol handler hands the packet to the IP handler via a function call.
The IP handler after doing the necessary table look-ups, determines which network interface to send it to and generates the software interrupt to the packet driver.
The packet driver then services the interrupt and places the packet on the wire.
Conversely, after the packet reaches the destination, the packet has to traverse the following path through the kernel before it reaches the user-level where it is time stamped:

The packet driver services the interrupt generated by the NIC and hands off the packet to the IP layer.
The IP layer after doing the necessary table lookups determines that the packet is destined locally, and hands the packet to the transport-layer.
The transport-layer does the necessary error checking and hands the packet to the user-level process. One or more context swaps may occur between the transport-layer handing the packet to the user process and the process actually time stamping the packet.

Since there is a cost associated with each of one these steps, the time-stamp on the test packet will differ from the wire-time both at the source and the destination. This difference is, however, very small on machines with fast processors, especially when the load on the machine is low. More importantly, as long the difference in the wire-times and the actual time-stamp is constant and measurable, that difference can be compensated for, after the measurement takes place.

Consider the case when the measurement machine is measuring the performance across a small number of paths. In this case:

The number of processes on the measurement machine is very low. It is unlikely that a context switch occurs between the packet arriving and the user-level process receiving the packet and time stamping the packet
The network traffic is low, reducing the likelihood that the packets are waiting in the network queues to be serviced by the protocol handler software.
In such conditions, the user-level software might yield acceptable accuracy. It is possible to carefully calibrate the measurement software to determine the constant difference between the wire-time and the actual time-stamps. So the measurement could be adjusted for this.

As the load increases, however, accuracy will suffer. In this case:

It is likely that one or more context switches occur between the packet arriving at the network interface and the user-level process time stamping the packet.
As there is large amount of test-traffic, the packets might wait in the network queues to be serviced by the protocol handler software.
It is extremely hard to measure the amount of time each individual test-packet might spend waiting the queues. This together with the variable delay introduced by the context switches means that the test-packets will have timestamps that are arbitrarily different from the wire-times.

Appendix E: Errors and uncertainties related to delay measurements

This section is an extract of RFC 2679, Chapter 3.7 [26]

E.1 Errors or uncertainties related to Clocks

The uncertainty in a measurement of one-way delay is related, in part, to uncertainties in the clocks of the Src and Dst hosts. In the following, we refer to the clock used to measure when the packet was sent from Src as the source clock, we refer to the clock used to measure when the packet was received by Dst as the destination clock, we refer to the observed time when the packet was sent by the source clock as Tsource, and the observed time when the packet was received by the destination clock as Tdest. Alluding to the notions of synchronization, accuracy, resolution, and skew mentioned in the Introduction, we note the following:

- Any error in the synchronization between the source clock and the destination clock will contribute to error in the delay measurement. We say that the source clock and the destination clock have a synchronization error of Tsynch if the source clock is Tsynch ahead of the destination clock. Thus, if we know the value of Tsynch exactly, we could correct for clock synchronization by adding Tsynch to the uncorrected value of Tdest-Tsource.

- The accuracy of a clock is important only in identifying the time at which a given delay was measured. Accuracy, per se, has no importance to the accuracy of the measurement of delay. When computing delays, we are interested only in the differences between clock values, not the values themselves.

- The resolution of a clock adds to uncertainty about any time measured with it. Thus, if the source clock has a resolution of 10 msec, then this adds 10 msec of uncertainty to any time value measured with it. We will denote the resolution of the source clock and the destination clock as Rsource and Rdest, respectively.
- The skew of a clock is not so much an additional issue as it is a realization of the fact that Tsynch is itself a function of time. Thus, if we attempt to measure or to bound Tsynch, this needs to be done periodically. Over some periods of time, this function can be approximated as a linear function plus some higher order terms; in these cases, one option is to use knowledge of the linear component to correct the clock. Using this correction, the residual Tsynch is made smaller, but remains a source of uncertainty that must be accounted for. We use the function Esynch(t) to denote an upper bound on the uncertainty in synchronization. Thus, |Tsynch(t)| <= Esynch(t).

Taking these items together, we note that naive computation Tdest-Tsource will be off by Tsynch(t) +/- (Rsource + Rdest). Using the notion of Esynch(t), we note that these clock-related problems introduce a total uncertainty of Esynch(t)+ Rsource + Rdest. This estimate of total clock-related uncertainty should be included in the error/uncertainty analysis of any measurement implementation.

E.2 Errors or uncertainties related to Wire-time vs. Host-time

As we have defined one-way delay, we would like to measure the time between when the test packet leaves the network interface of Src and when it (completely) arrives at the network interface of Dst, and we refer to these as "wire times." If the timings are themselves performed by software on Src and Dst, however, then this software can only directly measure the time between when Src grabs a timestamp just prior to sending the test packet and when Dst grabs a timestamp just after having received the test packet, and we refer to these two points as "host times".

To the extent that the difference between wire time and host time is accurately known, this knowledge can be used to correct for host time measurements and the corrected value more accurately estimates the desired (wire time) metric.

To the extent, however, that the difference between wire time and host time is uncertain, this uncertainty must be accounted for in an analysis of a given measurement method. We denote by Hsource an upper bound on the uncertainty in the difference between wire time and host time on the Src host, and similarly define Hdest for the Dst host. We then note that these problems introduce a total uncertainty of Hsource+Hdest. This estimate of total wire-vs-host uncertainty should be included in the error/uncertainty analysis of any measurement implementation.

Appendix F: References

[1] Hall, B. (2001). Beej's Guide To Network Programming, Using Internet Sockets, Revision 2.1.0. http://www.ecst.csuchico.edu/~beej/guide/net/bgnet.pdf (January 1st 2003).

[2] Cisco Systems. (2002). Diffserv Frequently Asked Questions. http://www.cisco.com/warp/public/cc/pd/iosw/ioft/ioqo/tech/dffs_qa.htm (January 1st 2003).

[3] Stepehson, A. (1999). Tech Tutorial: Diffserv and MPLS: A Quality Choice. http://www.networkmagazine.com/article/DCM20000502S0020 (January 1st 2003).

[4] Kalidindi, S. (1998). OWDP Implementation, v1.0, STR-002.

[5] Reijs, V. (2002). Deliverable D9.7. Testing of Traffic Measurement Tools. Geant GN1. IST-2000-26417. http://www.dante.net/geant/public-deliverables/GEA-02-079v2.pdf (January 1st 2003).

[6] Okazawa, H., Machizawa, A., Nakagawa, S., Kitaguchi, Y., Asami, T., Ito, A. (2001). Advanced NTP Synchronization Device for Internet Monitoring Tools. http://www.isoc.org/isoc/conferences/inet/01/CD_proceedings/T42/inet2001.html (January 1st 2003).

[7] Unknown Author. NTP8 Network Time Server Specification. http://www.brandywinecomm.com/literature/bwc_Ntp_NTP8.pdf (January 1st 2003).

[8] Purdue University. Cerias NTP Service. http://ntp.cerias.purdue.edu (January 1st 2003).

[9] Advanced Network & Services. Surveyor. http://www.advanced.org/surveyor/ (December 2nd 2002).

[10] Mills, D.L. (2001). NTP Performance Analysis. http://www.eecis.udel.edu/~mills/database/brief/new/new.pdf (January 1st 2003).

[11] Mills, D.L. Precision Time Synchronization. (2002). http://www.eecis.udel.edu/~mills/database/brief/precise/precise.pdf (January 1st 2003).

[12] Tierney, B., Johnston, W., Crowley, B., Hoo, G., Brooks, C., Gunter, D. (1998). The NetLogger Methodology for High Performance Distributed Systems Performance Analysis. http://citeseer.nj.nec.com/538230.html (January 1st 2003).

[13] Luckie, M. J., McGregor, A.J., Braun, H. (2001). Towards Improving Packet Probing Techniques. http://wand.cs.waikato.ac.nz/wand/publications/imw2001-probing.pdf (January 1st 2003).

[14] GPSClock.com. Tick and Tock, Public Stratum One NTP Servers. http://www.gpsclock.com/synch.shtml (January 1st 2003).

[15] Ixiacom. (2000). Timestamp Synchronization Between Distributed Chassis' Rev. 0.1. http://www.ixiacom.com/pdfs/whitepapers/TimestampSynchronizationPaper.pdf (January 1st 2003).

[16] Smotlacha, V. (2001). Measurement of Time Servers. http://www.cesnet.cz/doc/techzpravy/2001/18/ntprep.pdf (January 1st 2003).

[17] Reijs, V. (2002). Perceived quantitative quality of applications. http://www.heanet.ie/Heanet/projects/nat_infrastruct/perceived.html (January 1st 2003).

[18] Smotlacha, V. (2001). QoS Oriented Measurement in IP Networks. http://www.cesnet.cz/doc/techzpravy/2001/17/qosmeasure.pdf (January 1st 2003).

[19] Ubik, S. (2001). Low -Cost Precise QoS Measurement Tool. http://staff.cesnet.cz/~ubik/publications/2001/qosplot.pdf (January 1st 2003).

[20] Veitch, D., Pasztor, A. (2001). A Precision Infrastructure for Active Probing. http://www.emulab.ee.mu.oz.au/~darryl/probe1.ps.gz (January 1st 2003).

[20] Reijs, V. (2001). QoS monitoring and SLS auditing. http://www.heanet.ie/Heanet/projects/nat_infrastruct/qosmonitoringtf-ngn.html (January 1st 2003).

[21] Internet 2. (2001).Internet 2 QoS Working Group charter. http://www.internet2.edu/qos/wg/apps/appsQoS-charter.html (January 1st 2003).

[22] Padhye, J. (1998). Modeling TCP throughput: A simple model and its empirical validation http://www.acm.org/sigcomm/sigcomm98/tp/abs_25.html (January 1st 2003).

[23] Reijs V. (2001). Calculating TCP goodput http://www.heanet.ie/Heanet/projects/nat_infrastruct/tcpcalculations.htm (January 1st 2003).

[23] Almes, G. (1993). A One-way Packet Loss metric for IPPM. RFC 2680. http://www.advanced.org/IPPM/docs/rfc2680.txt (January 1st 2003).

[24] Koodli, R., Ravikanth, R. (2002). One-way loss pattern sample metrics. RFC 3357. http://www.ietf.org/rfc/rfc3357.txt (January 1st 2003).

[25] Chimento, P. (2002). IP Packet Delay Variation metric for IPPM. http://www.ietf.org/internet-drafts/draft-ietf-ippm-ipdv-09.txt (January 1st 2003).

[26] Almes, G. (1999). A One-way Delay metric for IPPM. RFC 2679. http://www.ietf.org/rfc/rfc2679.txt (January 1st 2003).

[27] Campanella, M., Chivalier, P., Sevasti, A., Simar, N. (2001) Quality of Service definition (SEQUIN, IST-1999-20841). http://www.dante.net/tf-ngn/SEQ-D2.1.pdf (January 1st 2003).

[28] Jacobson, V. Traceroute. (1988). ftp://ftp.ee.lbl.gov/pub/traceroute.tar.Z (January 1st 2003).

[29] Cisco Systems. (2002). Cisco IOS Configuration Tasks. http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/120newft/120limit/120xa/120xa1/fw1700.htm (January 1st 2003).

[30] Cisco Systems. (2002). Improving Security on Cisco Routers. http://www.cisco.com/warp/public/707/21.html#source-routing (January 1st 2003).

[31] Savage, S. (1999). Sting: a TCP-based Network Measurement Tool, http://www.cs.washington.edu/homes/savage/sting (January 1st 2003).

[32] Muuss, M. (1983). Ping. http://www.ping127001.com/ping.shar (January 2nd 2003).

[33] Mockapetris, P. (1987). Domain Names – Concepts and Facilities. RFC 1034. http://www.ietf.org/rfc/rfc1034.txt (January 6th 2003).

[34] Ripe. (2002). RIPE NCC Test Traffic Measurements. http://www.ripe.net/ripencc/mem-services/ttm/#Intro (January 6th 2003).

[35] Kalidindi, S. (1998). Techniques to Scale the Surveyor Infrastructure. STR-003. Advanced Network & Services.

[36] Paxson, V., Almes, G., Mahdavi, J., Mathis, M. (1998). Framework for IP Performance Metrics. RFC 2330. http://www.ietf.org/rfc/rfc2330.txt (January 8th 2003).

[37] Stevens, R. W., (2000). Programmieren von UNIX-Netzwerken. 2. Auflage. Carl Hanser Verlag.

[38] McGregor, A.J. (2002). IP Measurement Protocol. http://moat.nlanr.net/Papers/PAM02-IPMP.pdf (January 9th 2003).