Qosium Concepts

Some network measurement technology concepts are extensive and challenging to understand from a couple of lines of explanation. These articles delve deeper into the most common concepts that you may run across while using Qosium.

Table of Contents

1. Understanding Measurement Topology ... 4
1.1. To Start with ... 4
1.2. Typical Topology Models ... 4
1.2.1. End-to-End ... 5
1.2.2. One End-Point ... 6
1.2.3. On Traffic Path (in an Active Network Component) ... 7
1.2.4. On Traffic Path (in a Passive Network Component) ... 8
1.3. Network Interface Selection ... 9
1.4. Dealing with Tunnels ... 10
1.5. Direction of Traffic and Senders ... 10
2. Packet Filters in Qosium ... 11
2.1. Filtering in General ... 11
2.2. Filtering in Qosium ... 12
2.3. Additional Information ... 12
2.4. Pcap Filter Syntax Reference ... 13
2.4.1. Pcap Filter Syntax ... 13
2.4.1.1. Primitives ... 13
2.4.1.1.1. Host ... 13
2.4.1.1.2. Ether ... 13
2.4.1.1.3. Net & Mask ... 14
2.4.1.1.4. Port ... 14
2.4.1.1.5. Packet Size ... 15
2.4.1.1.6. Protocol ... 15
2.4.1.1.7. Broadcast ... 16
2.4.1.1.8. WLAN ... 16
2.4.1.1.9. VLAN ... 17
2.4.1.1.10. Multiprotocol Label Switching (MPLS) ... 17
2.4.1.1.11. Point-to-Point Protocol over Ethernet (PPPoE) ... 18
2.4.1.2. Relation Expression ... 18
2.4.1.3. Combining Primitives ... 19
2.4.2. Examples ... 19
2.4.2.1. Basic and Common Scenarios ... 19
2.4.2.2. PROFINET over Ethernet ... 20
2.4.2.3. Odd and Even Port Numbers ... 20
2.4.3. Special Cases ... 21
2.4.3.1. Sometimes the Order of Primitives Matter ... 21
2.4.3.2. Curious Filtering Cases ... 21
3. Passive QoS Measurement ... 22
3.1. Where Do We Need Passive QoS Measurement? ... 22
3.2. Passive Measurement in Practice ... 23
4. Quality of Experience ... 23
4.1. QoE Basics ... 23
4.2. The MOS Scale ... 23
4.3. QoE Models ... 24
5. Quality of Service ... 24
5.1. What is QoS and Why Is It Important ... 24
5.2. Typical QoS Statistics ... 25

6. Time Synchronization 26
6.1. NTP 26
6.2. PTP 26
6.3. The Use of NTP and PTP in Practise 27
6.4. GNSS 27
7. Positioning 27
7.1. Qosium Probe Parameterization 27
7.1.1. Activating Position Collection 28
7.2. Manual Position 28
8. Glossary 29

1. Understanding Measurement Topology

Where to install measurement agents and measurement controllers, what can be measured, how to set the parameters, etc. That is what understanding measurement topology is all about. It is one of the key things regarding QoS measurements.

1.1. To Start with

Measurement topology is something that deals with almost everything related to the Qosium measurement setup, including at least the following:

- The traffic to be measured and its directions
- Measurement nodes, their network interfaces, and other network devices involved
- Measurement software installations
- Measurement parameters

Thus, mastering the measurement topology is essential. To get started with Qosium measurements, consider the following:

- What application or service is the interesting one? Or are there, perhaps, multiple interesting flows or all traffic at some point in the network?
  - This is the key to everything in the measurement setup.
  - This also directly affects the packet filter to be set.
- Are you interested in full QoS results, or are traffic statistics (or just packet captures) enough?
  - Full QoS results mean that there will be two measurement points required.
- Where does the interesting traffic flow? What is the path or place of your interest for measuring this traffic?
  - Set the measurement points in such a way that the interesting path or place will be included. If just possible, install Qosium Probes directly to these points. If not, install them as near as possible. Another way is to mirror the traffic to be measured to an external device on which Qosium Probe runs.
- How are the Qosium Probe installations selected for the measurement located from the perspective of the application traffic you intend to measure (end-points, middle-points, external devices)?
  - Set the **placement parameters** accordingly.
- Through which network interfaces the interesting traffic flows in the measurement points?
  - Set the capture interfaces of the measurement points accordingly.
- If a two-point measurement, is there a *NAT* somewhere between the measurement points (a two-point measurement)?
  - If so, this requires special attention in parameterization.

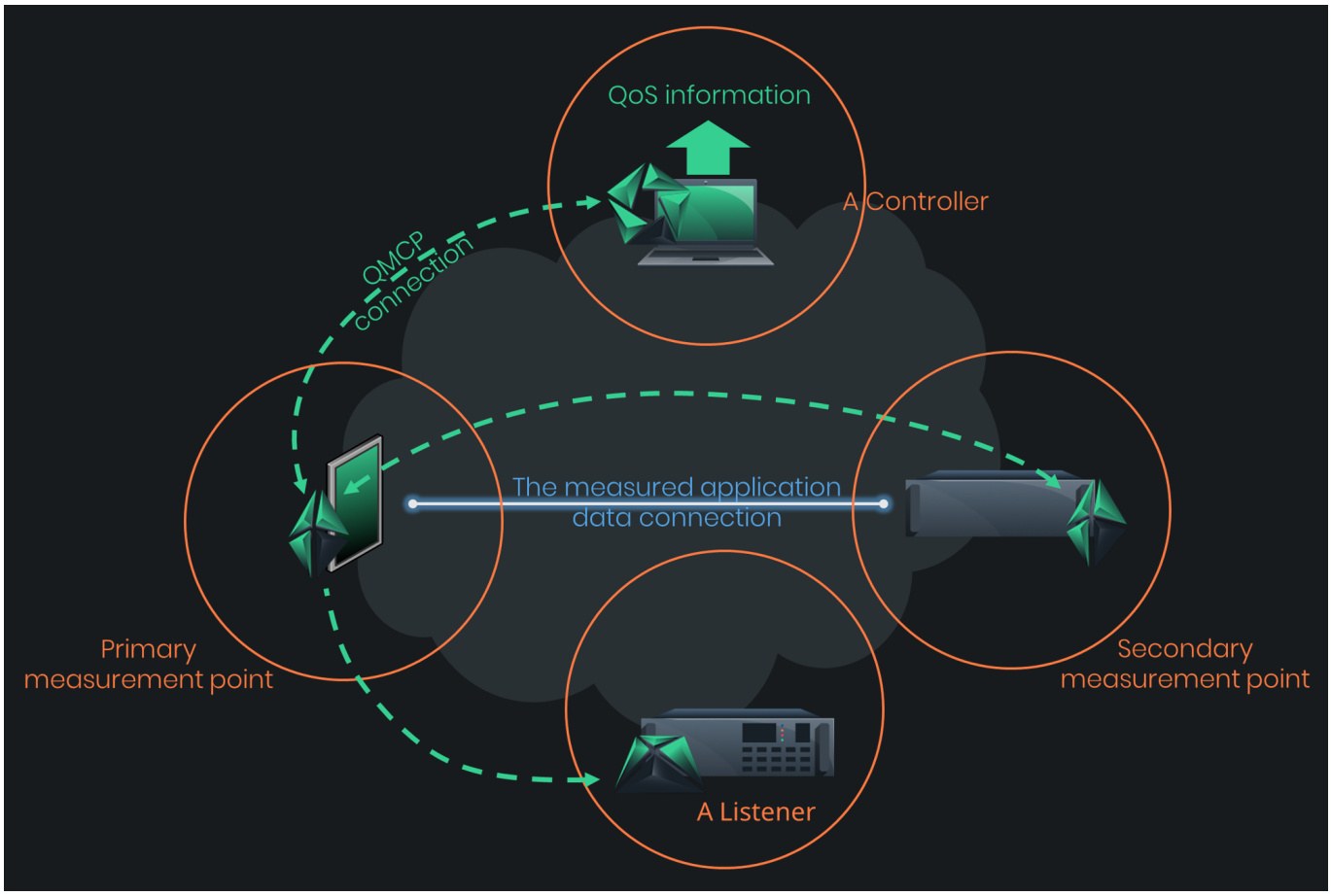
In the next section, we go through typical topology models.

1.2. Typical Topology Models

This section presents some general topology types. Please note that often, in reality, the topologies can be much more complex. However, even complex topologies typically resemble one of these at a logical level. In the presented measurement topologies, we focus only on two-point measurement cases.

A general two-point Qosium measurement with the *QMCP* flows, including a listener, is shown in the figure

below. You can select the measurement controller's location (e.g., Qosium Scope, Scopemon) quite freely. However, you should still consider the QMCP traffic flow orientation: they are chained, as seen in the figure below. The controller does not communicate directly with the Secondary measurement point but through the Primary. The optimal place in terms of minimizing the overhead is to run the controller in the Primary measurement point device. However, the QMCP connection between the controller and the Primary measurement point is very light-weight when measuring only average results and flow results. That is why the controller can very well be used remotely in a separate device, as illustrated in the figure.

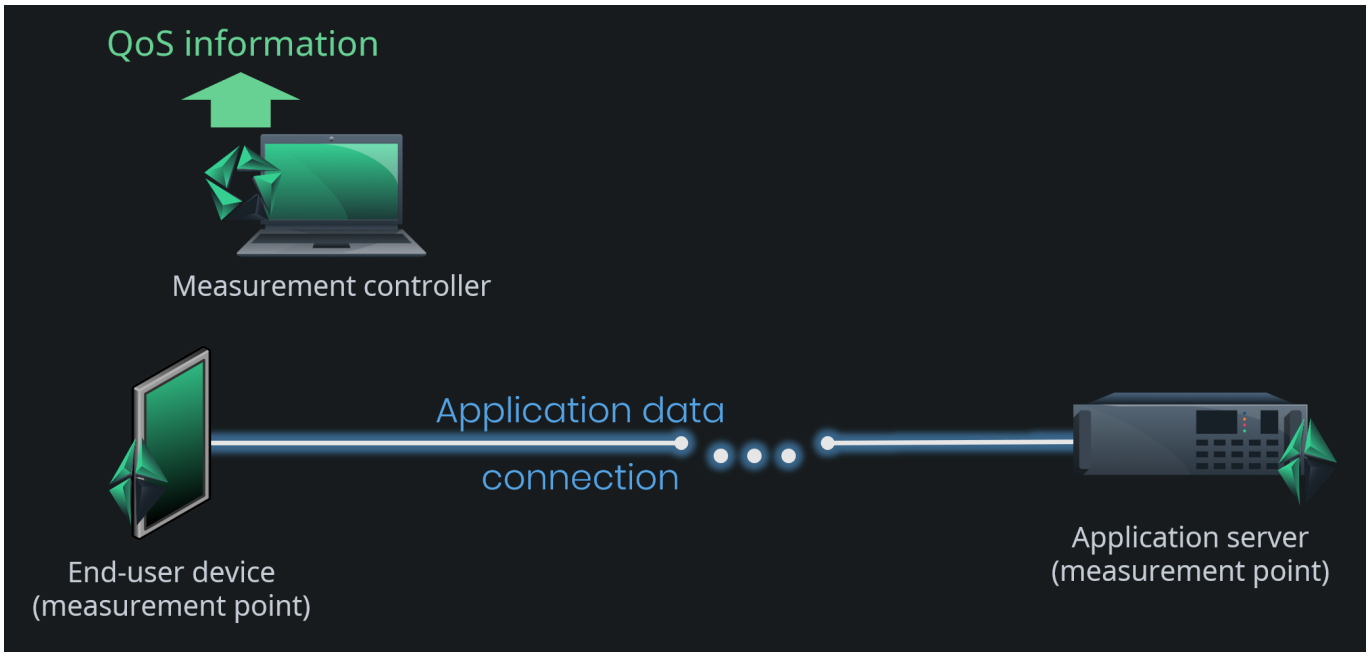


1.2.1. End-to-End

Are you interested in how your connected service works from the perspective of an end-user? The end-to-end topology reveals that.

This topology is the simplest one, and it is very common. The communicating end devices also act as measurement points. Thus, the results give the most accurate view of how the measured interesting application traffic performs end to end, i.e., the QoS level the network path between the devices gives to the application.

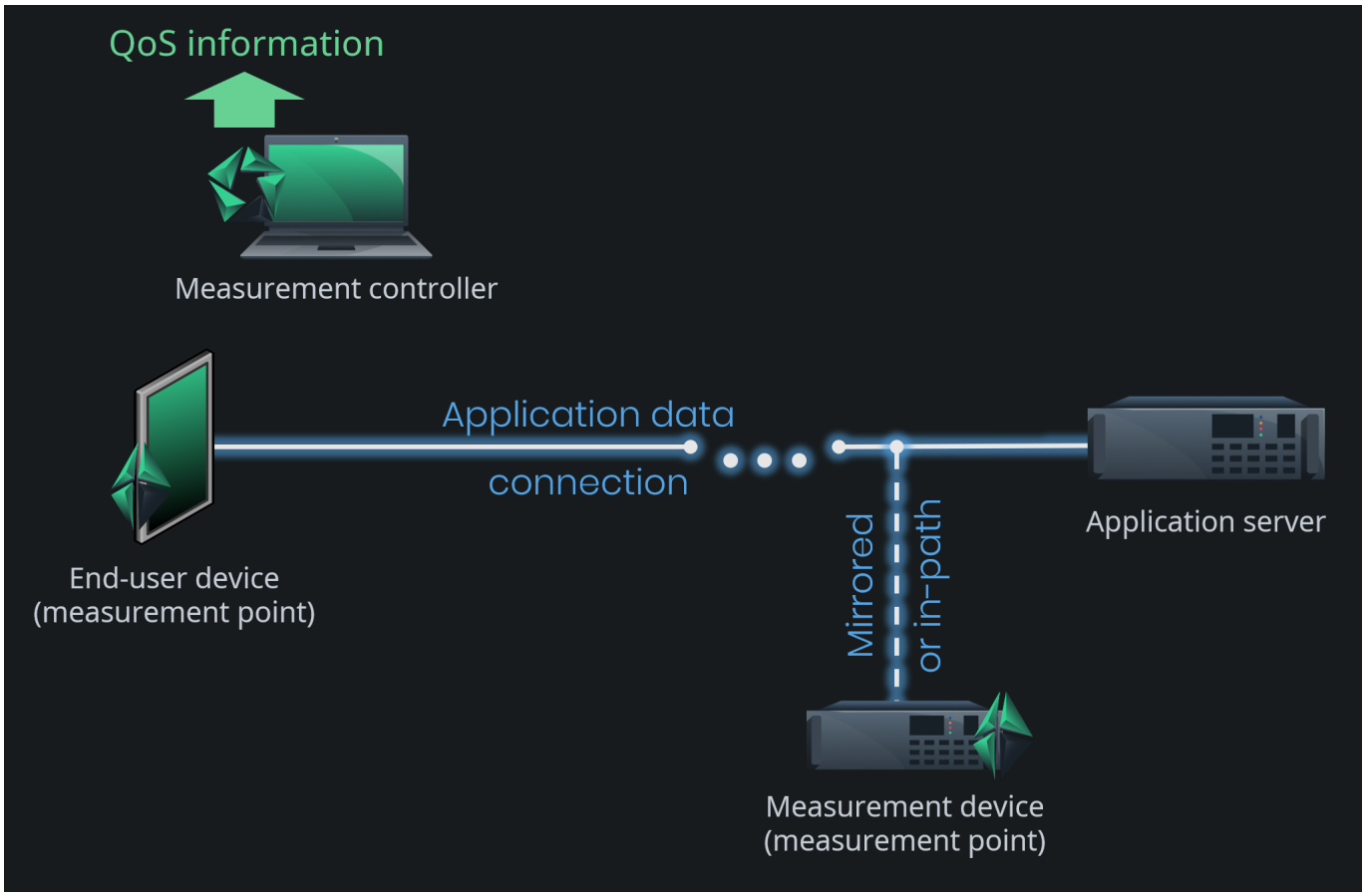
Parameterizing such measurement is straightforward. Just tell Qosium that the measurement nodes are at endpoints, and Qosium calculates almost all the other parameters automatically. Even the packet filter is automatized. The automatic filter will include all the traffic between the end devices. If you are interested only in specific traffic flows, tune the filter manually.



1.2.2. One End-Point

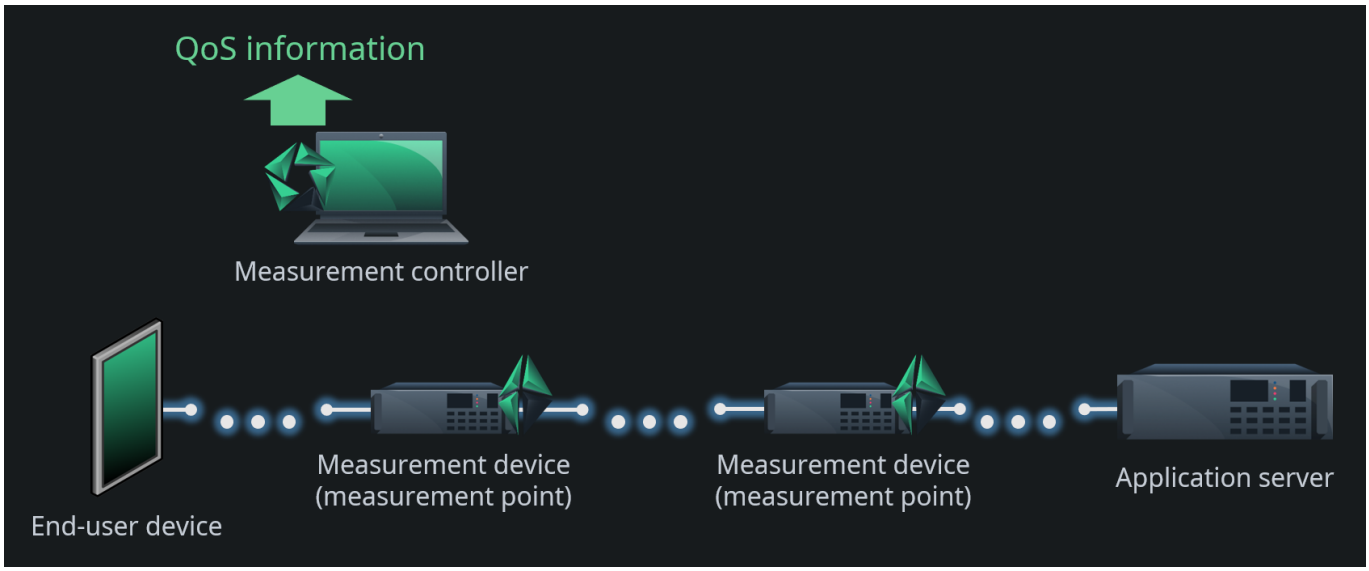
Sometimes it is not possible to install Qosium Probe in the end devices on both ends. Also, it might be that you are interested in the performance of your own network part, not the whole end-to-end part. As a result, we get a measurement topology, where one measurement point is in an end device, but the other is in the middle of the application traffic path. In some cases, the other point is not directly in the path but located in an external device, to which the traffic of your interest is mirrored for measurement purposes.

This measurement topology is still sufficiently easy to parameterize. The key difference, when compared to the end-to-end topology, is that automatic filtering is not possible. Thus, you must always set the packet filter manually in this topology option.



1.2.3. On Traffic Path (in an Active Network Component)

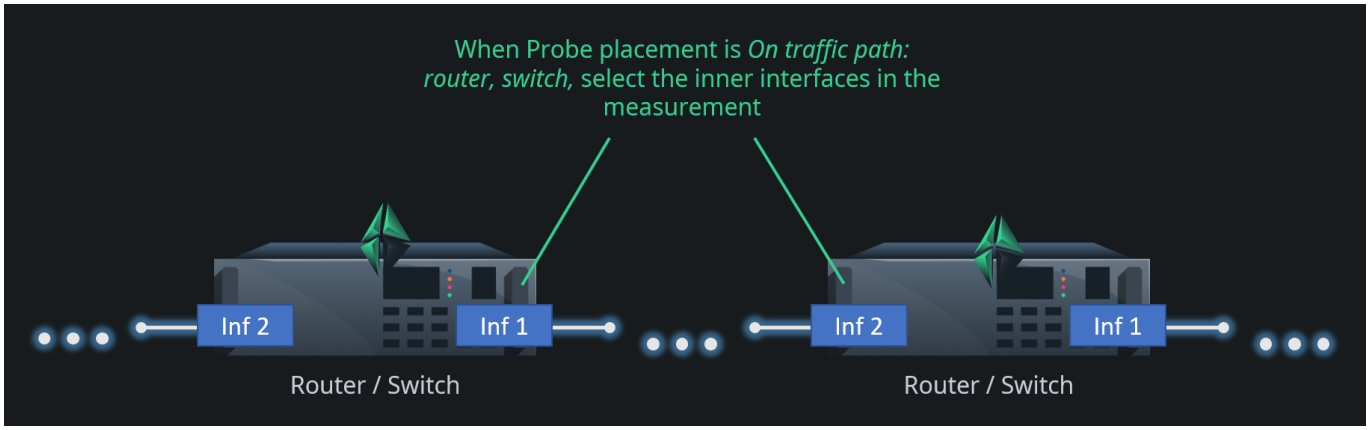
What if you are interested only in the quality of some specific part somewhere in the network? In this case, none of the measurement points are located in end devices but within the network path between the communicating devices. When Qosium Probes are installed on traffic path in an active network component, parameterization is still straightforward. By an active network component, we mean a device that participates in traffic handling actively, e.g., a router or a switch. Parameterization is similar to that of the One end-point topology. The main difference is in the placement parameters, naturally.



In this scenario, you need to pay some attention to the devices: how they operate and how they are

connected. For example, a network bridge is considered as a passive device, and parameterized accordingly. The reason is that MAC addresses are used to define the flow directions in this mode, but it does not work for passive devices that just let the traffic through untouched like a network bridge does.

Also, be careful when selecting the network interfaces in this kind of a scenario. You need to select the *inner* interfaces between the measurement points as shown in the figure below. Again, this has to do with MAC addressing. If you, however, want to measure over the devices, i.e., to use the *outer* interfaces, e.g., to include the switching/routing delay in the measurement, you need to consider the scenario as your measurement point would be a passive network component.

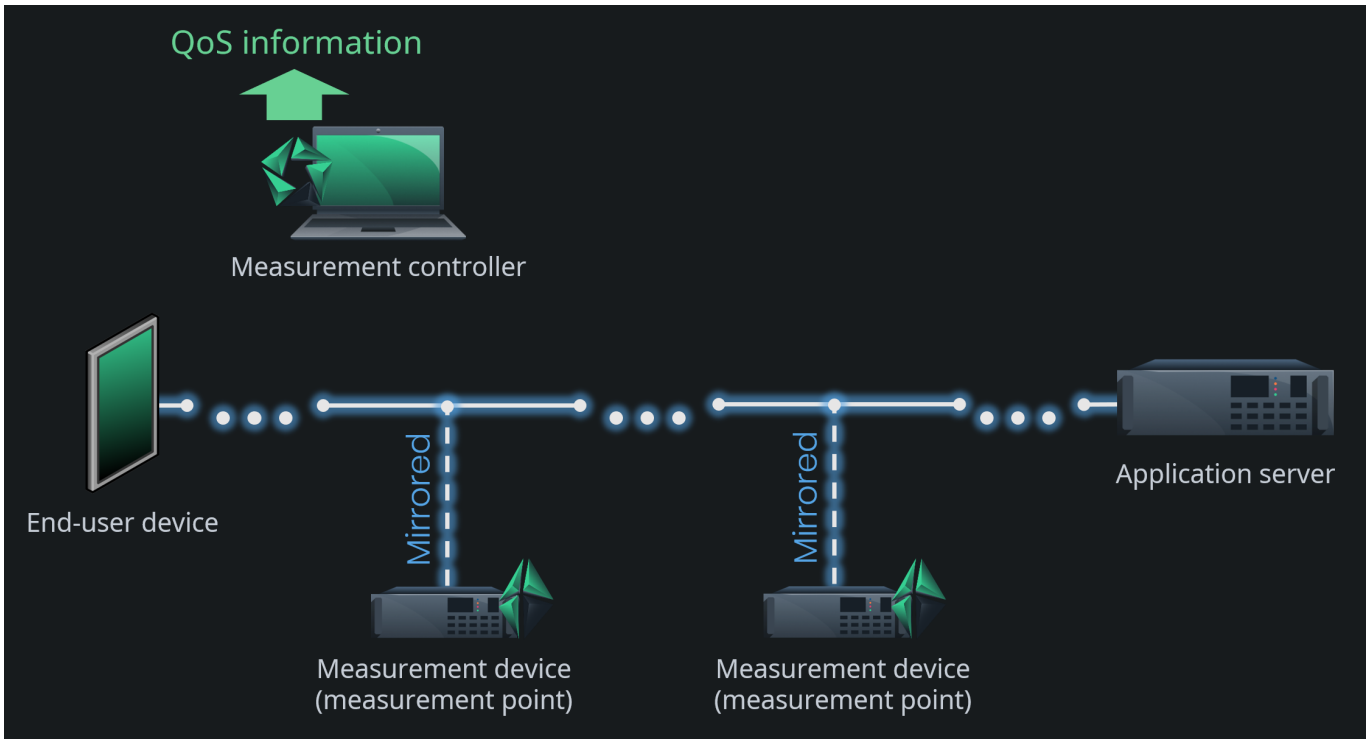


If the traffic or the network interfaces are such that there is no L2 addressing, you need to set the Sender parameters manually

1.2.4. On Traffic Path (in a Passive Network Component)

Consider a measurement topology like the previous one, but where Qosium Probes are installed on passive network devices. By this, we mean devices that do not actively modify traffic but try to let traffic through unmodified. The most obvious example of this kind of setup is a network tap that passively copies network traffic and forwards it to a separate measurement device. Another example is using port mirroring of a switch to forward the measured traffic to an external machine. A switch itself is an active network component, but now the measurement is not run in the switch. In addition, most network bridges fall into this category. Straightforward as such, but regarding passive measurement, the problem is that there is no longer accurate information available considering the direction of the traffic. Therefore, you need to give the measurement system hints considering the direction of the traffic. To do this in Qosium, set the Sender address parameters manually in this case. Also, you cannot use the automatic packet filter in this topology.

This is the most laborious basic topology in terms of parameterization. Should there be a *NAT* between the measurement points, you get a couple of parameter options more to do, but that is the most complex parameterization you can ever get in Qosium.

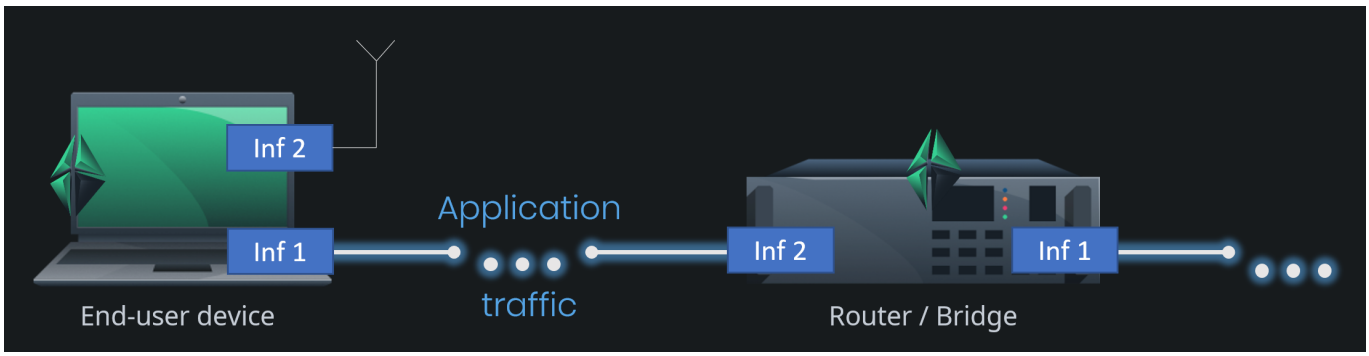


1.3. Network Interface Selection

Modern devices typically integrate multiple network interfaces. Consequently, it is essential to select the correct ones for the measurement. Choosing an incorrect interface is one of the most common mistakes in the measurement setup.

Consider the figure below. If you wish to measure the network path between the end-user device and the router, you need to select interface 1 in the end-user device and interface 2 in the router. By selecting interface 2 in the end-user device will show all packets coming from the router as lost. Besides, all packets received by the router are calculated as *SINF* as their sending is not registered in the end-user device. If you select interface 1 in the router, the measurement will succeed, but the measured QoS includes the routing process performance. Of course, if that is what you want, then the interface selection is correct.

Since Qosium is a multi-thread solution, you can carry out interesting measurements by using only a single Probe. Consider if you perform a two-point measurement in the figure's router. While the same Qosium Probe acts as the Primary and the Secondary measurement point, select different interfaces to them. For example, set the Primary measurement point to listen interface 2 and the Secondary to interface 1. What you get now with this kind of setup is the internal routing performance of the device. Moreover, since both of the measurement points are in the same device, the clock synchronization is ideal. You will be able to measure very accurate delay performance up to the microsecond level. Only the router's internal process priorities cause some minor inaccuracies.



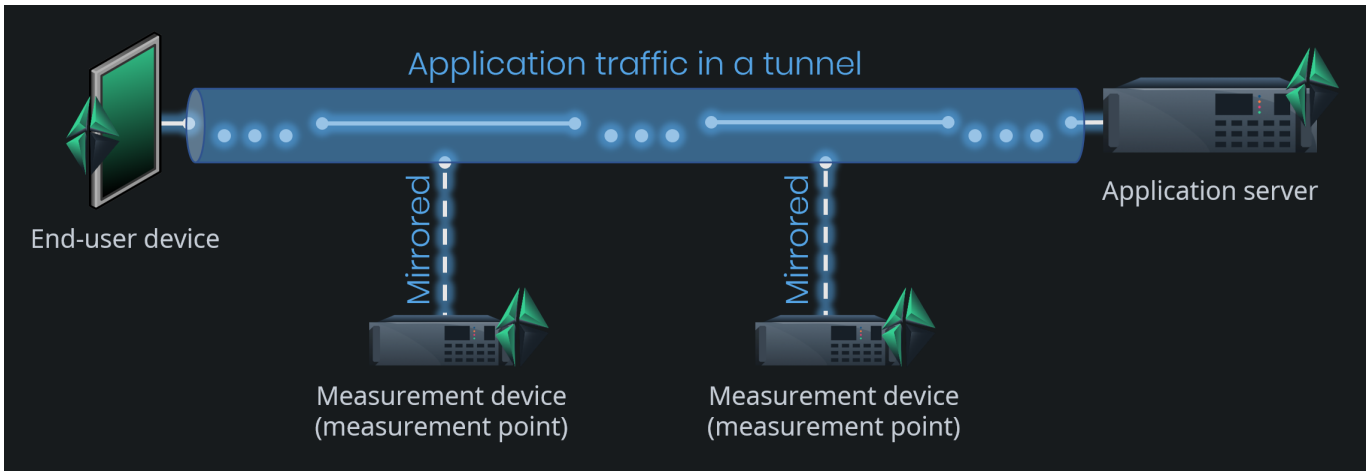
1.4. Dealing with Tunnels

A two-point passive QoS measurement works only if the packet context in both ends matches. Consequently, if there is a tunnel, pipe, or similar within the network path, the measurement point selection needs some attention.

Consider the figure below: the application traffic is flowing inside a tunnel (e.g., *VPN* between an end-user device and an application server. A successful QoS measurement can be made end-to-end since both of the end-points lie outside the tunnel. Also, a successful measurement can be carried out between the off-path external measurement points since they both see only the tunneled traffic. However, a measurement from the end-user device to one of the external measurement points, generally, cannot be made since the packet context does not match. The end-user devices see the packets as they are, but the external measurement points see tunneled packets, often encrypted.

There are some exceptions, however, of tunneling protocols over which Qosium measurements are possible:

- *MPLS*
  - Qosium is equipped with functionalities to dig the original traffic flows inside a non-encrypted MPLS tunnel.
  - However, some Pcap-versions can have problems to filter inside an MPLS tunnel. Without filtering, a QoS measurement will very likely fail. Of course, it is possible to perform manual low-level filtering based on individual protocol fields, but this gets easily laborious.
- *GTP*
  - Kaitotek has made a Linux kernel module that can extract the traffic inside a GTP tunnel, enabling it to be measured as such.
  - Please ask Kaitotek support or sales about the QGTPR module.

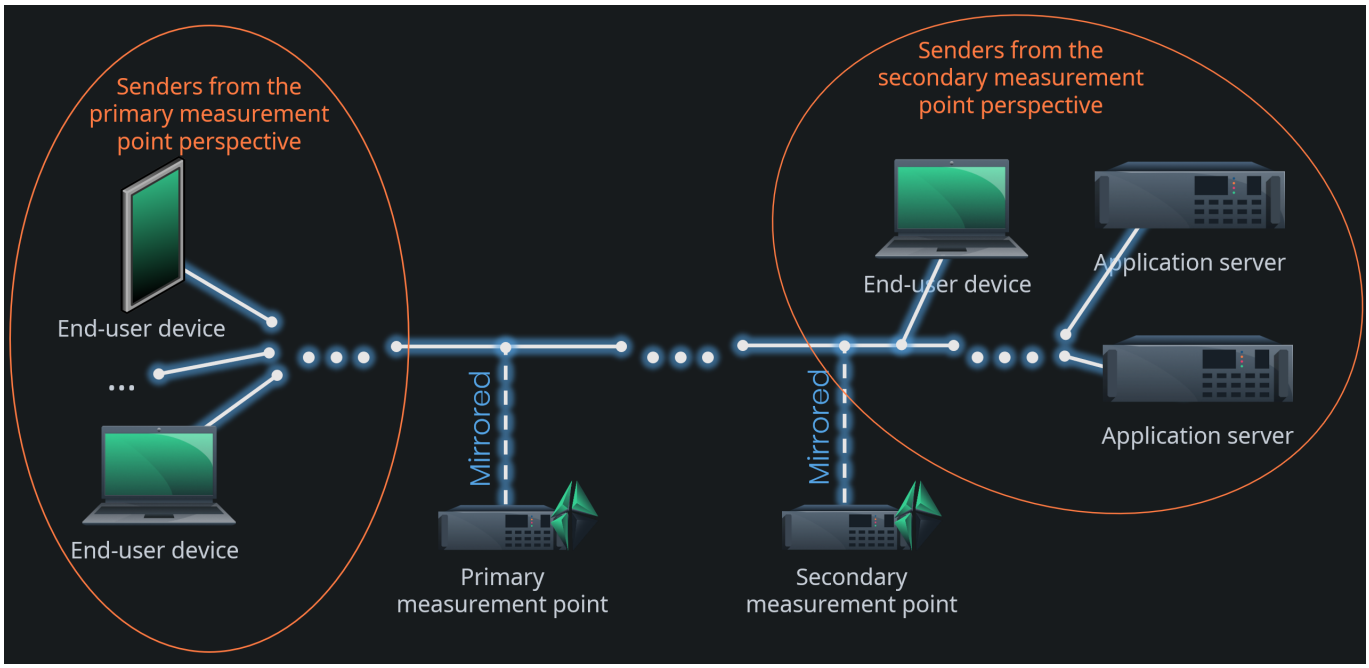


1.5. Direction of Traffic and Senders

Consider the end-to-end measurement topology: which way is the traffic flowing? Determining that is very easy since the measurement points are the same devices that generate the traffic: just compare device addresses to the traffic addresses. But, what about in a topology where the measured traffic is mirrored to external measurement devices? You could think that it is obvious. Just look at the topology. But, Qosium does not see the topology as we do. Instead, Qosium Probes only see the device in which they run and now a single NIC through which all the traffic is coming in. The traffic is not originated, nor meant to that node, so there is no information left to tell which way the traffic was flowing in its original point of capture.

Because of the two-point measurement, the measured one-way delay could be used to tell the direction, but as there can be clock synchronization errors, it cannot be trusted to cover all cases.

Therefore, there are special cases where you need to tell Qosium by extra parameters which way the traffic is flowing. This is done by the **Sender parameters**. You tell Qosium Probe, who, from its perspective, are *the senders*, i.e., the devices who send traffic towards the other measurement point. See the figure below. If, for example, one end-user device on the left-hand side of the figure is communicating with an application server on the right, the end-user is *a sender* from the Primary measurement point perspective. Then, the application server is a sender from the Secondary measurement point perspective. That you need to tell Qosium with parameters. It is typically enough to determine the senders only in one measurement point and then use *the inverse definition* (in Qosium Scope: *According to primary/secondary Probe*) in the other.



Qosium allows setting senders as individual addresses or as address range by using a mask. Senders can be set independently for IPv4, IPv6, and MAC addresses.

2. Packet Filters in Qosium

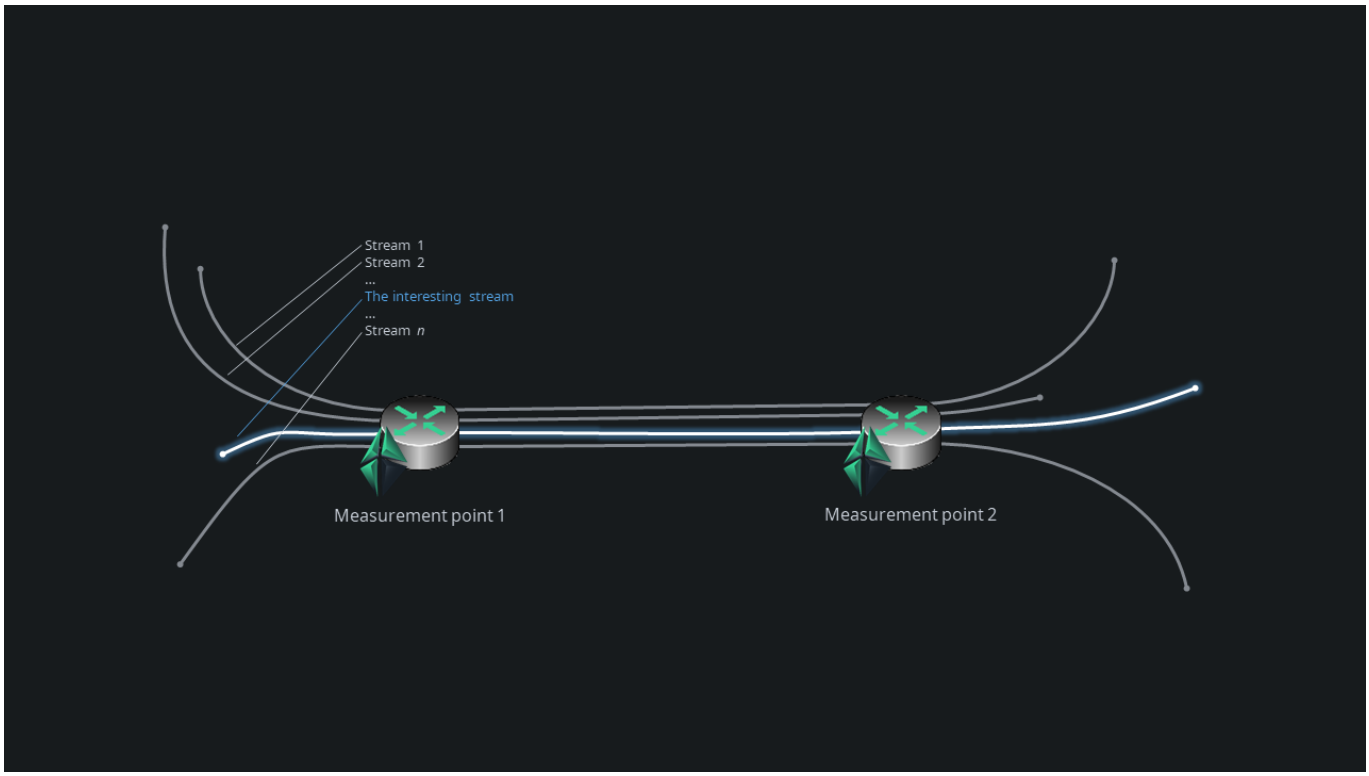
Packet filter is one of the essential concepts in passive measurement. An active measurement tool pushes a predetermined data stream into the network, so the traffic to be measured is always known. In passive measurement, on the other hand, existing traffic is measured, so filters are needed to define what part, in particular, of the total traffic is the interesting one to be taken into the measurement.

2.1. Filtering in General

In passive measurement, existing traffic in the network is being measured. By default, when tapping into a network interface, all the traffic traveling there is shown. Sometimes that is what is desired. However, often you may wish to examine only a specific part of the total traffic.

Consider the figure below, which depicts two measurement points through which several independent traffic streams flow, originating from different applications/services. If we were, for example, interested only in the QoS of a particular single flow between these two measurement points, we would need to filter out all the other flows during measurement. This enables us to measure the QoS solely from the perspective of that interesting traffic flow. Similarly, we could be interested in multiple flows of a specific kind, which would require another kind of filter. This is what filtering in the context of passive QoS

measurements is all about.



2.2. Filtering in Qosium

Qosium has some automation in filtering. For example, in an end-to-end measurement scenario, Qosium will, by default, assume that all traffic between the two points is included in the measurement and generate an automatic filter accordingly. In many measurement cases, that is enough. If, however, you are interested only in a specific part of the total traffic, tell that to Qosium by defining *a manual filter*. All Qosium measurement controllers allow defining a filter.

Defining a filter in Qosium follows the well-known Pcap syntax. A typical end-to-end filter could look like this:

```
ip and host 192.168.0.10 and host 192.168.0.101
```

This instructs Qosium to measure all IPv4 based traffic between hosts 192.168.0.10 and 192.168.0.101. If, for example, the interesting part would be a VoIP-stream traveling between these two machines, we need to focus the filter. Let us assume that the VoIP software uses UDP port 7000 on one end and UDP port 7001 on the other. Now we can define the filter as

```
ip and host 192.168.0.10 and host 192.168.0.101 and udp and port 7000 and port 7001
```

This is now a so-called *strict filter*, as there can be only one two-way stream that fits the filter. The previous filter was an example of a *loose filter* as many potential streams can be caught with that.

2.3. Additional Information

Writing filters in Qosium is mostly straightforward. An up-to-date full instruction of the Pcap syntax is given here. See our article on Pcap syntax instructions, which includes keywords, functionalities, and examples

typically needed when using Qosium.

> Qosium's packet filter is a capture filter, so the traffic filtered out cannot be added in the measurement later.

2.4. Pcap Filter Syntax Reference

Qosium uses Pcap syntax for defining manual packet filters. To master at least the very basic filtering cases is very useful when working with Qosium. This section provides information about general Pcap filter syntax that is often relevant to Qosium.

2.4.1. Pcap Filter Syntax

While some information is given next, a full syntax reference is found here.

2.4.1.1. Primitives

Important Pcap filter primitives are the following:

2.4.1.1.1. Host

Syntax	Condition
`dst host <host>`	IPv4/v6 destination field of the packet is `host`, which may be either an address or a name
`src host <host>`	IPv4/v6 source field of the packet is `host`
`host <host>`	IPv4/v6 source or destination of the packet is `host`

Any of the above host expressions can be prepended with the keywords `ip`, `arp`, `rarp`, or `ip6`, as in `ip host <host>`, which is equivalent to `ether proto \ip and host host`. If the host is a name with multiple IP addresses, each address will be checked for a match.

2.4.1.1.2. Ether

Syntax	Condition
`ether dst <ehost>`	Ethernet destination address is `ehost`, which may be either a name from /etc/ethers or a number
`ether src <ehost>`	Ethernet source address is `ehost`
`ether host <ehost>`	Ethernet source or destination address is `ehost`

True if the packet used the host as a gateway. I.e., the Ethernet source or destination address was host, but neither the IP source nor the IP destination was the host. The host must be a name and must be found both by the machine's host-name-to-IP-address resolution mechanisms (hostname file, DNS, NIS, etc.) and by the machine's host-name-to-Ethernet-address resolution mechanism (/etc/ethers, etc.). (An equivalent expression is ether host ehost and not host host, which can be used with either names or numbers for host/ehost.) This syntax does not work in an IPv6-enabled configuration at this moment.

```
gateway <host>
```

2.4.1.1.3. Net & Mask

Syntax	Condition
`dst net <net>`	IPv4/v6 destination address of the packet has a network number of `net`
`src net <net>`	IPv4/v6 source address of the packet has a network number of `net`
`net <net>`	IPv4/v6 source or destination address of the packet has a network number of `net`
`net <net>mask <netmask>`	IPv4 address matches net with the specific netmask. May be qualified with `src` or `dst`. Notice that this syntax is not valid for IPv6 net
`net <net>/<len>`	IPv4/v6 address matches `net`with a netmask `len` bits wide. May be qualified with `src` or `dst`

Net may be either a name from the network's database (/etc/networks, etc.) or a network number. An IPv4 network number can be written as a dotted quad (e.g., 192.168.1.0), dotted triple (e.g., 192.168.1), dotted pair (e.g., 172.16), or a single number (e.g., 10); the netmask is 255.255.255.255 for a dotted quad (which means that it's really a host match), 255.255.255.0 for a dotted triple, 255.255.0.0 for a dotted pair, or 255.0.0.0 for a single number. For IPv6 addresses, the network can only be defined by using the network number and the mask length. Thus, for example, filter

```
net fe80:1234:5678:9abc:0000:0000:0000:0000/64
```

includes all traffic containing the IPv6 addresses is in the range:
fe80:1234:5678:9abc:0000:0000:0000:0000 - fe80:1234:5678:9abc:ffff:ffff:ffff:ffff.

2.4.1.1.4. Port

Syntax	Condition
`dst port <port>`	Packet has a destination port value of `port`
`src port <port>`	Packet has a source port value of `port`
`port <port>`	Either the source or destination port of the packet is `port`
`dst portrange <port1>-<port2>`	Packet has a destination port value between `port1` and `port2`
`src portrange <port1>-<port2>`	Packet has a source port value between `port1` and `port2`
`portrange <port1>-<port2>`	Packet has a source or a destination port value between `port1` and `port2`

True if the packet is IPv4/IPv6 TCP, IPv4/IPv6 UDP, or IPv4/IPv6 SCTP, in some systems, and has a destination port value of port. The port can be a number or a name used in /etc/services. If a name is used, both the port number and protocol are checked. If a number or ambiguous name is used, only the port

number is checked (e.g., dst port 513 will print both TCP/login traffic and UDP/who traffic, and port domain will print both TCP/domain and UDP/domain traffic).

Any of the above port or port range expressions can be prepended with the keywords `tcp or`udp`. For example,`tcp src port `matches only TCP packets whose source port is`.

2.4.1.1.5. Packet Size

Syntax	Condition
`less <length>`	Packet has a length less than or equal to `<length>`. This is equivalent to `len <= <length>`
`greater <length>`	Packet has a length greater than or equal to `<length>`. This is equivalent to `len >= <length>`

2.4.1.1.6. Protocol

Syntax	Condition
`tcp`	Short for `proto tcp`
`udp`	Short for `proto udp`
`icmp`	Short for `proto icmp`
`ip proto <protocol>`	Packet is an IPv4 packet of protocol type `<protocol>`
`ip protochain <protocol>`	Packet is IPv4 packet, and contains protocol header with type `<protocol>` in its protocol header chain
`ip6 proto <protocol>`	Packet is an IPv6 packet of protocol type `<protocol>`
`ip6 protochain <protocol>`	Packet is IPv6 packet, and contains protocol header with type `<protocol>` in its protocol header chain
`ether proto <protocol>`	Packet is of ether type `<protocol>`, where protocol can be `ip`, `ip6`, `arp`, `rarp`, `atalk`, `aarp`, `decnet`, `iso`, `stp`, `ipx`, `netbeui`, `lat`, `moprc`, `mopdl`. Note that not all applications using currently know how to parse these protocols
`iso proto <protocol>`	Packet is an OSI packet of protocol `<protocol>`. Protocol can be a number or one of the names clnp, esis, or isis. Abbreviations for IS-IS PDU types are: l1, l2, iih, lsp, snp, csnp, psnp

The protocol can be a number or, e.g., one of the names `icmp`, `icmp6`, `igmp`, `igrp`, `pim`, `ah`, `esp`, `vrrp`, `udp`, or `tcp`. Note that the identifiers `tcp`, `udp`, and `icmp` are also keywords and must be escaped via backslash (), which is \ in the C-shell. Note that this primitive does not chase the protocol header chain.

`ip6 protochain 6` matches any IPv6 packet with TCP protocol header in the protocol header chain. The packet may contain, for example, an authentication header, routing header, or hop-by-hop option header between IPv6 header and TCP header. The BPF code emitted by this primitive is complex and cannot be optimized by the BPF optimizer code, which can be somewhat slow.

Ethernet protocol can be a number or one of the names ip, ip6, arp, rarp, atalk, aarp, decnet, sca, lat, mopdl, moprc, iso, stp, ipx, or netbeui. Note these identifiers are also keywords and must be escaped via

backslash ().

In the case of FDDI (e.g., `fddi protocol arp`), Token Ring (e.g., `tr protocol arp`), and IEEE 802.11 wireless LANs (e.g., `wlan protocol arp`), for most of those protocols, the protocol identification comes from the 802.2 Logical Link Control (LLC) header, which is usually layered on top of the FDDI, Token Ring, or 802.11 headers.

When filtering for most protocol identifiers on FDDI, Token Ring, or 802.11, the filter checks only the protocol ID field of an LLC header in so-called SNAP format with an Organizational Unit Identifier (OUI) of 0x000000, for encapsulated Ethernet; it doesn't check whether the packet is in SNAP format with an OUI of 0x000000. The exceptions are:

- `iso` - the filter checks the DSAP (Destination Service Access Point) and SSAP (Source Service Access Point) fields of the LLC header
- `stp` and `netbeui` - the filter checks the DSAP of the LLC header
- `talk` - the filter checks for a SNAP-format packet with an OUI of 0x080007 and the AppleTalk etype

In the case of Ethernet, the filter checks the Ethernet type field for most of those protocols. The exceptions are:

- iso, stp, and netbeui - the filter checks for an 802.3 frame and then checks the LLC header as it does for FDDI, Token Ring, and 802.11
- atalk - the filter checks both for the AppleTalk etype in an Ethernet frame and for a SNAP-format packet as it does for FDDI, Token Ring, and 802.11
- aarp - the filter checks for the AppleTalk ARP etype in either an Ethernet frame or an 802.2 SNAP frame with an OUI of 0x000000
- ipx - the filter checks for the IPX etype in an Ethernet frame, the IPX DSAP in the LLC header, the 802.3-with-no-LLC-header encapsulation of IPX, and the IPX etype in a SNAP frame

2.4.1.1.7. Broadcast

Syntax	Condition
`ether broadcast`	Packet is an Ethernet broadcast packet. The ether keyword is optional
`ip broadcast`	Packet is an IPv4 broadcast packet
`ether multicast`	Packet is an Ethernet multicast packet. The ether keyword is optional. This is shorthand for `ether[0] & 1 != 0`
`ip multicast`	Packet is an IPv4 multicast packet
`ip6 multicast`	Packet is an IPv6 multicast packet

`ip broadcast` checks for both the all-zeros and all-ones broadcast conventions and looks up the subnet mask on the interface on which the capture is being done.
If the subnet mask of the interface on which the capture is being done is not available, either because the interface on which capture is being done has no netmask or because the capture is being done on the Linux *any* interface, which can capture on more than one interface, this check will not work correctly.

2.4.1.1.8. WLAN

Syntax	Condition
`wlan addr1 <ehost>`	First IEEE 802.11 address is `<ehost>`
`wlan addr2 <ehost>`	Second IEEE 802.11 address, if present, is `<ehost>`. The second address field is used in all frames except for CTS (Clear To Send) and ACK (Acknowledgment) control frames
`wlan addr3 <ehost>`	Third IEEE 802.11 address, if present, is `<ehost>`. The third address field is used in management and data frames, but not in control frames
`wlan addr4 <ehost>`	Fourth IEEE 802.11 address, if present, is `<ehost>`. The fourth address field is only used for WDS (Wireless Distribution System) frames
`dir <dir`	IEEE 802.11 frame direction matches the specified dir. Valid directions are *nods*, *tods*, *fromds*, *dstods*, or a *snumeric* value
`type <wlan_type>`	IEEE 802.11 frame type matches the specified `<wlan_type>`. Valid WLAN types are *mgt*, *ctl* and *data*
`subtype <wlan_subtype>`	IEEE 802.11 frame subtype matches the specified `<wlan_subtype>` and frame has the type to which the specified WLAN subtype belongs
`type <wlan_type> subtype <wlan_subtype>`	IEEE 802.11 frame type matches the specified `<wlan_type>`and frame subtype matches the specified `<wlan_subtype>`. If the specified wlan_type is `mgt`, then valid wlan_subtypes are: *assoc-req*, *assoc-resp*, *reassoc-req*, *reassoc-resp*, *probe-req*, *probe-resp*, *beacon*, *atim*, *disassoc*, *auth*, and *deauth*. If the specified wlan_type is `ctl`, then valid wlan_subtypes are: *ps-poll*, *rts*, *cts*, *ack*, *cf-end*, and *cf-end-ack*. If the specified wlan_type is `data`, then valid wlan_subtypes are *data*, *data-cf-ack*, *data-cf-poll*, *data-cf-ack-poll*, *null*, *cf-ack*, *cf-poll*, *cf-ack-poll*, *qos-data*, *qos-data-cf-ack*, *qos-data-cf-poll*, *qos-data-cf-ack-poll*, *qos*, *qos-cf-poll* and *qos-cf-ack-poll*

2.4.1.1.9. VLAN

Virtual LAN (VLAN) IEEE 802.1Q tagged packets can be filtered with the `vlan` keyword. Filter `vlan <vlan_id>` yields packets that have the corresponding VLAN ID. Note that the first `vlan` keyword encountered in expression changes the decoding offsets for the remainder of the expression on the assumption that the packet is a VLAN packet. The `vlan <vlan_id>` expression may be used more than once to filter on VLAN hierarchies. Each use of that expression increments the filter offsets by 4.

For example, to filter on VLAN 200 encapsulated within VLAN 100:

```
vlan 100 && vlan 200
```

To filter IPv4 protocols encapsulated in VLAN 300 encapsulated within any higher order VLAN:

```
vlan && vlan 300 && ip
```

2.4.1.1.10. Multiprotocol Label Switching (MPLS)

Syntax	Condition
`mpls <label_num>`	Packet is an MPLS packet. If `<label_num>` is specified, the packet must have the corresponding label_num. Note that the first `mpls` keyword encountered in expression changes the decoding offsets for the remainder of the expression on the assumption that the packet is an MPLS-encapsulated IP packet. This expression may be used more than once to filter on MPLS hierarchies. Each use of that expression increments the filter offsets by 4

For example, filter packets with an outer label of 100000 and an inner label of 1024:

```
mpls 100000 && mpls 1024
```

Filter packets to or from 192.9.200.1 with an inner label of 1024 and any outer label:

```
mpls && mpls 1024 && host 192.9.200.1
```

2.4.1.1.11. Point-to-Point Protocol over Ethernet (PPPoE)

Syntax	Condition
`pppoed`	Packet is a PPP-over-Ethernet Discovery packet (Ethernet type 0x8863)
`pppoes`	Packet is a PPP-over-Ethernet Session packet (Ethernet type 0x8864). Note that the first pppoes keyword encountered in expression changes the decoding offsets for the remainder of expression on the assumption that the packet is a PPPoE session packet

For example, filter IPv4 protocols encapsulated in PPPoE:

```
pppoes && ip
```

2.4.1.2. Relation Expression

```
expr relop <expr>
```

True if the relation holds, where `relop` is one of `>`, `<`, `>=`, `<=`, `=`, `!=`, and `expr` is an arithmetic expression composed of integer constants (expressed in standard C syntax), the normal binary operators `+`, `-`, `*`, `/`, `&`, `|`, `<<`, `>>`, a length operator, and special packet data accessors. Note that all comparisons are unsigned, so that, for example, `0x80000000` and `0xffffffff` are `> 0`. To access data inside the packet, use the following syntax:

```
<proto> [ <expr> : <size> ]
```

Proto can be is one of `ether`, `fddi`, `tr`, `wlan`, `ppp`, `slip`, `link`, `ip`, `arp`, `rarp`, `tcp`, `udp`, `icmp`, `ip6` or `radio`, and indicates the protocol layer for the index operation. (`ether`, `fddi`, `wlan`, `tr`, `ppp`, `slip`, and `link` all refer to the link layer. `radio` refers to the *radio header* added to some 802.11 captures.) Note that `tcp`, `udp`, and other upper-layer protocol types only apply to IPv4, not IPv6 (this will be fixed in the future). The byte offset, relative to the indicated protocol layer, is given by `expr`. Size is optional and indicates the number of

bytes in the field of interest; it can be either one, two, or four and defaults to one. The length operator, indicated by the keyword `len`, gives the length of the packet.

For example, `ether[0] & 1 != 0` catches all multicast traffic. The `&`-sign means bit-wise masking, so the above expression basically checks the first byte's last bit 1. The expression `ip[0] & 0xf != 5` catches all IPv4 packets with options. The expression `ip[6:2] & 0x1fff = 0` catches only unfragmented IPv4 datagrams and frag zero of fragmented IPv4 datagrams. This check is implicitly applied to the `tcp` and `udp` index operations. For instance, `tcp[0]` always means the first byte of the TCP header and never means the first byte of an intervening fragment.

Some offsets and field values may be expressed as names rather than as numeric values. The following protocol header field offsets are available: `icmptype` (ICMP type field), `icmpcode` (ICMP code field), and `tcpflags` (TCP flags field).

The following ICMP type field values are available: `icmp-echoreply`, `icmp-unreach`, `icmp-sourcequench`, `icmp-redirect`, `icmp-echo`, `icmp-routeradvert`, `icmp-routersolicit`, `icmp-timxceed`, `icmp-paramprob`, `icmp-tstamp`, `icmp-tstampreply`, `icmp-ireq`, `icmp-ire-qreply`, `icmp-maskreq`, `icmp-maskreply`.

The following TCP flags field values are available: `tcp-fin`, `tcp-syn`, `tcp-rst`, `tcp-push`, `tcp-ack`, `tcp-urg`.

2.4.1.3. Combining Primitives

Primitives may be combined using:

- A parenthesized group of primitives and operators (parentheses are special to the Shell and must be escaped)
- Negation (`!` or `not`)
- Concatenation (`&&` or `and`)
- Alternation (`||` or `or`)

Negation has the highest precedence. Alternation and concatenation have equal precedence and associate left to right. Note that explicit and tokens, not juxtaposition, are now required for concatenation.

If an identifier is given without a keyword, the most recent keyword is assumed. For example, `not host vs and ace` is short for `not host vs and host ace`, which should not be confused with `not ( host vs or ace )`.

2.4.2. Examples

2.4.2.1. Basic and Common Scenarios

Next, some elementary examples of the use of Pcap syntax are given:

To get all traffic seen in the interface, enter an empty filter.

To get all IPv4 based traffic is:

```
ip
```

To get all traffic that involves my host (called myhost):

```
host myhost
```

To select all IPv4 traffic between 192.168.1.1 and 192.168.1.7:

```
ip host 192.168.1.1 and host 192.168.1.7
```

To select all IPv4 traffic between 192.168.1.1 and any host except 192.168.1.7:

```
ip host 192.168.1.1 and not host 192.168.1.7
```

To select all TCP traffic, including port 80 between 192.168.1.1 and 192.168.1.7:

```
ip host 192.168.1.1 and host 192.168.1.7 and tcp and port 80
```

To select all UDP traffic with even source ports between 192.168.1.1 and 192.168.1.7 or 192.168.1.6

```
ip and udp and (host 192.168.1.1 and (host 192.168.1.7) or (host 192.168.1.6)) and
(udp[0:2] & 1 = 0)
```

2.4.2.2. PROFINET over Ethernet

When running the PROFINET protocol in the real-time mode, it runs directly over Ethernet without IP. Currently, there is no automatic filtering for Ethernet packets, so one has to perform the filtering manually.

Assume that there are two hosts, *01:02:03:04:05:06* and *07:08:09:0a:0b:0c*, which communicate with PROFINET. If there is no other traffic traveling between the hosts, traffic between these hosts can be measured with a simple filter:

```
ether host 01:02:03:04:05:06 and ether host 07:08:09:0a:0b:0c
```

This will include all Ethernet traffic between the hosts. Thus, if there are, e.g., IP streams above Ethernet, those will also be included. When limiting the focus purely on PROFINET, this is done by a filter:

```
ether proto 0x8892
```

The reason for this kind of syntax is that PROFINET's protocol ID, or *Type* in Ethernet, is 8892 in hex. Pcap does not support PROFINET protocol directly. The filter above can be enough if one is sure that PROFINET communications take place only between the selected hosts. However, if there are more PROFINET streams, from the hosts to other hosts, the hosts' Ethernet addresses need to be included:

```
(ether host 01:02:03:04:05:06 and ether host 07:08:09:0a:0b:0c) and (ether proto 0x8892)
```

When using PROFINET, Virtual LANs are often used to improve QoS. This needs to be taken into account in the filter level as well, as discussed earlier. Thus, the final filter of this example will be in the form:

```
(ether host 01:02:03:04:05:06 and ether host 07:08:09:0a:0b:0c) and
((ether proto 0x8892) or (vlan and ether proto 0x8892))
```

2.4.2.3. Odd and Even Port Numbers

Capturing all UDP packets with an even-numbered port becomes topical when measuring RTP streams

without RTCP-messages. Typically, RTP uses even-numbered ports and RTCP odd-numbered ones.

We cannot use the port statement since it cannot be manipulated in the way we need. Instead, we need to dig the protocol's port fields. By masking the least significant bit, we can reach our goal. A filter `udp[0:2] & 1 = 0` will take the first two bytes bits into checking, i.e., the source port. It masks that with 1, so when this is 0, the port must be even-numbered. Thus, the above filter takes all packets whose source port has an even-numbered value. Similarly, a filter `udp[2:2] & 1 = 0` includes all packets whose destination port is even-numbered. Then again, the filter `udp[2:2] & 1 = 1` includes all packets whose destination port is odd-numbered.

2.4.3. Special Cases

2.4.3.1. Sometimes the Order of Primitives Matter

There are some things to notice as Pcap libraries perform packet filtering in a particular order. For example, if there is a *VLAN* tag in the Ethernet header of a packet, at least some versions of Pcap won't go inside the packet at all if not especially told to do so. Thus, the packet content cannot be reached by usual filtering. Instead, insert `vlan and …` in front of the manual packet filter, and the filter works again as expected.

In some special two-point measurement setups, there can be VLAN tag in one end of the measurement, while not in the other. In this case, you cannot set a simple filter, but instead, the case with and without VLAN must be dealt with separately (with `or` statement) in the filter. For example, if you wish to measure all IP traffic, the filter would become:

```
ip or (vlan and ip)
```

It is important to notice here that in Pcap (at least in most of the Pcap versions), the order of the `vlan` statement in the filter matters! More specifically, the non-VLAN tagged part of the filter must be informed before the VLAN-tagged part. Otherwise, you will see only the VLAN-tagged part regardless of what comes next in the filter. Thus, if we write our previous example filter in the format:

```
(vlan and ip) or ip
```

which is logically the same as above, it will not work.

A good article related to this topic is given here.

If the two measurement points have different VLAN-setup, i.e., the other has a VLAN tag the other does not, you can also use auto-filtering. Just set the filter for the Primary Probe end and then select suitable autofilter mode. Notice: the autofiltering modes are currently quite simple, so if you have a complex filter, this won't work.

2.4.3.2. Curious Filtering Cases

A peculiar filtering behavior, at least, in some versions of Pcap (involving libpcap and Npcap), has been detected. Consider that you have two identical devices that both have two network interfaces. One interface has a fixed IP address of 192.168.0.1. The other interface is used (now in this example) for communications, and they have IP addresses, per device, of 192.168.2.100 and 192.168.2.101. Thus, Qosium's original automatic end-to-end filtering would yield a filter of

`ip and (host 192.168.0.1 or host 192.168.2.100) and (host 192.168.0.1 or host 192.168.2.101)` .

As seen, it has some redundancy since the IP address 192.168.0.1 is there twice, but from a logical point of view, it should work. It should give all IPv4-based traffic between the hosts 192.168.2.100 and 192.168.2.101, but it does not! Instead, it lets in only directional flows as if the filter would have been of

form:

`ip and src host 192.168.2.100 and dst host 192.168.2.101`.

Thus, in general, if the filter is of form:

`ip and (host <IP 3> or host <IP 1>) and (host <IP 3> or host <IP 2>)`

, where the goal is to measure all IPv4-based traffic between <IP 1> and <IP 2>, it does not work as assumed. The causing factor is somehow related to the extra IP address (<IP 3>) being common to both sides of the filter. Instead, if the same filter is written in different order:

`ip and (host <IP 1> or host <IP 3>) and (host <IP 2> or host <IP 3>)`

, it works despite the fact that logically it should be the same as previous. Therefore, the order of primitives matters here. Then, of course, a filter:

`ip and (host <IP 1> or host <IP 3>) and (host <IP 2> or host <IP 4>)`

, works normally, as there is no common factor between the two sides of the filter that is, presumably, causing the issue. There is also a worse problem. Consider a filter of form:

`ip and (host <IP 1> or host <IP 2> or host <IP 4>) and (host <IP 1> or host <IP 3> or host <IP 4>)`

, or generally,

`ip and (host <IP n1> or host <IP m1> or host … or host <IP M> or host <IP n2>) and (host <IP n1> or host <IP k1> or host … or host <IP K> or host <IP n2>)`.

That kind of filter structure halts Pcap completely. The behavior has been detected, at least with some versions of libpcap. Npcap (at least from version 1.80 forward) does not halt, but it does not show any traffic either, even though, logically, it should.

The exact reasons for these detected behaviors of Pcap filtering are yet unknown to us. They could be some obvious normal consequences of the internal behavior of Pcap filter composition that we have missed or not.

Nevertheless, it shall be pointed out that none of the presented curious filtering cases are typically problematic: they are redundant, make no sense as such, and can be easily formed alternatively. When performing manual filtering, one will likely never face these issues. However, attention should be paid to automatic filtering. From Qosium Probe 1.9.2.0 onwards, these issues have been taken into consideration in the formation of the automatic filter. But, if you are building your own automatic monitoring setups with custom automatic parameterization, you also should pay attention to this.

3. Passive QoS Measurement

In short, passive QoS measurement is about measuring QoS statistics for real existing application traffic in the network. This is a significant difference to most other network performance measurement solutions available. Passive QoS measurement is the optimal way to measure and monitor network performance after commissioning.

3.1. Where Do We Need Passive QoS Measurement?

There are plenty of solutions and tools available for measuring and monitoring network performance and quality. However, most of them are active ones that don't work well when the network is in operational usage. *Active tools* determine QoS and/or QoE only for the artificial test traffic they generate in one end and measure at the other end of the network path of interest. The generated traffic can mimic some real application, or they are used to obtain the maximum throughput performance. These tools are very useful in

verifying newly deployed networks that they work as designed or benchmarking network throughputs with no mission-critical applications employed over them.

With *active measurement solutions*, however, you don't have visibility on how real existing applications experience the network performance. This is the case, although you try to simulate their traffic characteristics. By burdening operational networks with artificial test traffic, this can result in real applications experiencing degraded quality. It is especially important to know how real applications and users experience the network quality when you have connected applications and services with QoS demands, like latency constraints. This is where *passive network measurement solutions* comes into the play, namely to know how network connections serve real applications and respond to their QoS demands.

It is good to realize that active and passive measurement technologies are not competitors of each other. Instead, they are meant for different use cases, completing each other.

3.2. Passive Measurement in Practice

Let's take an example where there is live video stream application traffic in the network. If an active measurement tool is used to measure the performance, you get results for that artificially generated traffic stream only. However, the actual video stream you are interested in can perform differently. Besides, the active test traffic stream changes the measured system as it adds traffic to the measured network path. In the worst case, it congests the network path of interest. This means that you get incorrect results while interfering with the live video stream user. Passive measurement, on the contrary, evaluates the actual video stream and measures how well the network serves it from the QoS perspective. As a result, you get, for example, *delay*, *jitter*, *packet loss*, and *connection breaks* just for the desired real video stream over the measurement path of interest. Passive QoS measurement also enables you to estimate QoE with application-specific models to get an indication of how satisfied the end-user is for the video quality. When passive QoS measurement is carried out in real-time, some control information needs to be exchanged between the measurement points. The amount of this overhead, however, is often meaningless.

4. Quality of Experience

Quality of Experience (QoE) indicates how satisfied the user is for using the application/service. QoE is always an application-specific measure, and the actual result can vary from person to person. This section introduces how QoE can be estimated automatically and in real-time for a connected application without consulting the user.

4.1. QoE Basics

QoE is typically related to applications used over network connections. This is one main difference to User Experience (UX), which is much associated with, for example, how well the user interface matches user's expectations. Network connections bring impairments to the application traffic. However, QoE does not indicate only the quality of connection but includes the entire service. For example, in a video application, the used codec, picture resolution, compression rate, etc., are all aspects that affect how delighted the user is with the video quality. Although application configuration and settings affect QoE, when they are known, clearly, the most variation to user satisfaction comes from the network QoS, from delay, jitter, and packet loss.

4.2. The MOS Scale

The most widely used measure for QoE is *MOS*, 1-5. The value of 5 means excellent, non-impaired satisfaction, and the value of 1 that the application is not usable at all. We use the MOS scale also in Qosium due to its widespread utilization.

4.3. QoE Models

Qosium comes with QoE models that estimate user satisfaction. A scientifically proven technique, *PSQA*, can accurately estimate the quality as perceived by the application users. PSQA was originally developed for assessing multimedia applications. The model is based on a neural network technology trained with data acquired from actual user tests. Users are asked to assess a set of application samples, for example, video samples over different, pre-defined, network conditions. As PSQA uses real-time QoS statistics (e.g., delay, jitter, and packet loss) in the assessment process, careful preparation of the test sample material is essential.

Currently, Qosium has integrated models for a few VoIP codecs and an H.264 encoded video (SD, HD, and Full-HD resolutions). However, Kaitotek has experience in developing models for new applications and media codecs. Overall, the model development process has three steps:

- Create samples with different kind of impairments
- Collect a group of people, end-users, for the user test
- Create a neural network model

PSQA is not the optimal model for applications for which the end-user is not a human but, for example, a robot. Most robots are still deterministic what comes to their perception of network connection quality. The same robots perceive the application usability the same in similar conditions. Qosium also has a QoE model, which you can parameterize yourself. When you know the tolerated delay, jitter, packet loss, and/or connection break durations for your application(s), the generic model calculates you a single real-time value in the MOS scale. However, the estimation of the user experience is based purely on the settings values of yours. The generic model is known as *GQoSM*.

You can argue that is this generic model QoE. By the original QoE definition, it’s definitely not. However, if you have many different applications, developing PSQA models for each of them may be too heavy. Moreover, suppose you want to measure the quality of multiple applications simultaneously, but you would rather like to know the quality of the connection than the absolute user experience without following numerous QoS statistics separately. In that case, the generic model works better as you also likely already know the QoS demands of your application(s). Although the generic model is not based on user tests, the difference in the result value to user-test-based models is observed to be only fractions in many cases.

In summary, it depends much on the use case that do you need a very accurate estimation of how satisfied users truly are, for example, for voice and video applications. Or, could a single model that suits all your applications with different parameter settings but does not provide you with that accurate estimation be still sufficient? Whatever your need is, both of these methods are supported by Qosium.

5. Quality of Service

QoS, when measured in communications networks, indicates the overall performance of the connectivity. For measuring the network QoS, the most common QoS statistics include delay/latency, jitter (delay variation), available data rate, and packet loss. As statistics already indicate, QoS is always a measure carried out between two points of interest in the network. Qosium measures QoS from network traffic level, indicating the QoS experienced by applications.

“Regarding communications, the only thing that matters to the end-user, whether a human or a machine, is the connection quality.”

5.1. What is QoS and Why Is It Important

A communications system is at its best when the user doesn’t notice its existence while using networked applications and services. Thus, an ideal data communications system would have an unlimited data rate

and a constant undelayed and lossless delivery of data packets. In that case, you can think QoS to be indefinitely good. But, as we all know, this is a utopia, so we must be satisfied with something much less. That is not an issue, however, since the meaning of QoS is always application specific. It is enough that the QoS of a communications system is high enough for the applications and services to run smoothly. The requirements vary a lot: a communications path whose quality satisfies one application fully, can be a disaster to another.

For example, a VoIP conversation has very small data rate requirements, and also some packet losses are tolerated except in high compression ratio codecs. The delay starts to get annoying only after some 100 – 200 ms. In online gaming, in contrast, that delay level would already be far too much for games requiring fast reactivity. Then again, industrial automation applications typically require both very low delay and zero packet loss to work safely and efficiently. On the other hand, over-the-top streaming media services such as Netflix and YouTube demand throughput capacity rather than low delays and packet loss levels. While delay and packet loss will also slow down streaming services and bulk data transmission, content buffering in the end device takes mostly care of the sporadic QoS variation in the connectivity.

Is maximum throughput performance QoS? For bulk data transfer, it mostly is, but generally not: it represents only one feature of QoS. The tools that allow you to measure your connection's speed only tell how much capacity there's in reserve in your network connection at that time towards the test server. It may give you a hint that a bulk data transfer or a high-quality video stream runs smoothly. But it tells you little about how applications requiring low delay, low jitter, and low packet losses would work. A Ping test often accompanies maximum data speed measurements. It measures *RTT*, a representation of two-way joint delay, but it is only for that Ping application. Some other applications having completely different packet length profiles, packet rate variations, and protocols over the same connection likely experience different delays. Consider if your connection gives sporadic one-way delay spikes for one packet per thousand on average, a high-rate video stream contains these spikes every second. If we assume that Ping's data packets experience the same kind of spiky delay behavior, being not always the case, with standard settings, it takes hundreds of seconds, on average, to notice the spikes by Ping, and thousands to make a more deterministic perception.

In telecommunications terminology, you often see QoS mentioned in the context of data prioritization, where it refers to the QoS target rather than the realized QoS. For example, many wireless systems are equipped with QoS class definitions to manage different traffic flows differently. There are dedicated bearer definitions for different types of traffic and subscriptions in mobile networks. DiffServ is one commonly known standard for classifying and managing IP traffic. While these solutions attempt to provide the application traffic with satisfactory QoS, they cannot guarantee that. The classification and the following data prioritization can favor defined traffic types over the others, but they cannot battle, e.g., low radio link quality efficiently. In addition, the methods generally do not work end-to-end but just for a limited part of the total network path. Thus, even though these methods help, you still don't know how well it went, i.e., what kind of QoS did your applications get over the network path.

For end-users, often, the only thing that matters is that the application used works as it should. The application works when it gets the QoS it requires from the network. The application does not care about the underlying network technology. It can be wired or wireless, as long as delays, delay variation, and packet loss stay within tolerable boundaries and the realized data transfer rate is enough for the used application. Real-time QoS measurement enables you to monitor how well the network can serve your applications and how usable your applications are.

5.2. Typical QoS Statistics

The table below introduces the main QoS statistics. Application QoS demands determine which of them play the essential roles. As the statistics indicate, they are such that you always need to have a reference point over which they can be calculated, i.e., QoS is always determined for the path between two network points. For instance, calculating the delay for an IP packet necessitates that you know when that packet has been sent on the other end of the measurement path. As this is not simple to be carried out, the typical

way has been to measure RTT. It, however, cannot tell you what has the delay behavior been in the sent and receive directions, which can sometimes vary a lot. If the packet involved in the round-trip measurement is lost, you cannot know which direction caused this loss. Having one-way statistics helps you analyze network problems in more detail. In addition, as discussed earlier, RTT is typically an *active measurement*, where the traffic to be measured is particularly generated for the purpose. In order to measure real applications' QoS, *passive QoS measurement methods* are required.

Statistic	Description
Delay	Time difference between the time data is transmitted to the moment it is received by the other measurement point. Sometimes delay is also called *latency*.
Jitter	Time how much delay differs between sequential packets
Packet loss ratio	Ratio between successfully transmitted and lost packets on the measurement path
Connection break	Time duration from the moment a packet loss is detected to a moment packet is successfully transferred
Available data rate	Indicates how fast the connection can transfer data

6. Time Synchronization

When measuring one-way delay, clock synchronization between the measurement points is essential. Here you find some basic information on synchronizing clocks and finding the best solution for your needs.

Qosium relies on the system clock in a measurement point in most cases. For measuring one-way delay, this means that the clocks on the measurement points need to be synchronized for good results. Synchronization errors show up in the results. For example, with two-way data, the other direction's delay can be negative, and the delay to another direction displays high positive delays. Synchronization error can be detected as mirrored delay behavior between the send and received directions.

Below you find the most common synchronization methods with some typical accuracy levels. For more information on different synchronization methods, please reach out to Kaitotek.

6.1. NTP

NTP is the most common way to synchronize clocks over a network connection. It's supported practically by all computers with a network interface. NTP allows you to reach a few milliseconds accuracy at its best, but you need to be prepared to experience an inaccuracy of about 10 ms. Thus, NTP does not suit use cases well where you need to get very accurate delay results.

Essential with NTP synchronization is that you configure NTP to poll time frequently enough. For example, the NTP client (w32time) on Windows defaults to a configuration that polls time from a server rarely after the NTP client had determined it has reached a synchronization. Qosium installer for Windows helps you reconfigure this so that the clock is requested from the NTP server at a maximum of every 16 seconds despite the client's state. Overall, it is suggested to use maximum polling intervals between 8-30 seconds.

6.2. PTP

As NTP, *PTP* is a synchronization protocol to be used over a network. PTP is designed for local networks where the network devices need to be synchronized with high accuracy. For example, mobile networks rely on PTP.

PTP allows you to reach tens-to-hundreds microsecond accuracy. The better is the accuracy, the more stable is the network connection. For example, the synchronization accuracy is better in wired networks than over a wireless connection. However, reaching an accuracy level of hundreds of microseconds over a wireless connection is possible by tuning the PTP client settings not to adjust the clock too sensitively.

In addition to the stability of network connection and the configuration of a PTP client, timestamp methods supported by the operating system and the host hardware affect the obtainable accuracy. Timestamp methods come with tradeoffs, for example, between granularity, accuracy, precision, and efficiency.

There are a few open-source PTP implementations available for Linux. Some of them require, for example, hardware timestamping support from the network interface, but some of them work on basically all Linux-based systems with software timestamping. An example of a versatile open-source PTP implementation is PTPd. On Windows, natively included *w32time* is also possible to be utilized as a PTP client starting on Windows 10 and Windows Server 2019. You can run PTPd in a PTP master mode and also synchronize your Windows machine(s) from that over UDP for substantially improved clock synchronization compared to NTP. Contact Kaitotek support for more detailed instructions and scripts regarding PTPd and PTP on Windows.

6.3. The Use of NTP and PTP in Practise

Some instructions on parameterizing NTP and PTP for Linux-based systems are provided here.

Instructions for Windows, mainly for NTP, are provided here.

6.4. GNSS

Good absolute timing accuracy, about tens of microseconds, can be reached with *GNSS*. The key is a GNSS receiver with a *PPS* capability. A pulse is sent once a second, starting at the beginning of every new second. Another important aspect is to feed the host machine with the pulse signal. Old-good RS242 serial line communications provide high accuracy while, for example, USB loses the accuracy.

On Linux, the actual time synchronization is then carried out with NTP locally. On Windows, you need a separate driver for managing PPS signal with performance counters.

See here how to take GNSS into use in Linux-based systems.

7. Positioning

Qosium results are tied to time but can also be tied to location.

Qosium supports getting the position of the device in which the Qosium Probe measurement agent runs. When measuring wireless networks, this is valuable information to understand the network quality also in terms of position.

7.1. Qosium Probe Parameterization

Qosium Probe supports the following ways to get the position:

- On Linux, if you are using GPSD, Probe can connect directly to the GPSD API for real-time reading of the position.
- If you have a position available elsewhere, e.g., in an external GNSS receiver, you can send the location to Probe using NMEA-0183 (GGA) messages over UDP.

7.1.1. Activating Position Collection

Open the *QosiumProbe.ini* file. It is default located in `c:\Program Files\Qosium\` on Windows, `/opt/QosiumProbe/bin/` on Linux, and `/Applications/QosiumProbe.app/Contents/MacOS/` on macOS.

Find the *Positioning parameters* section and modify the *location_mode* value accordingly:

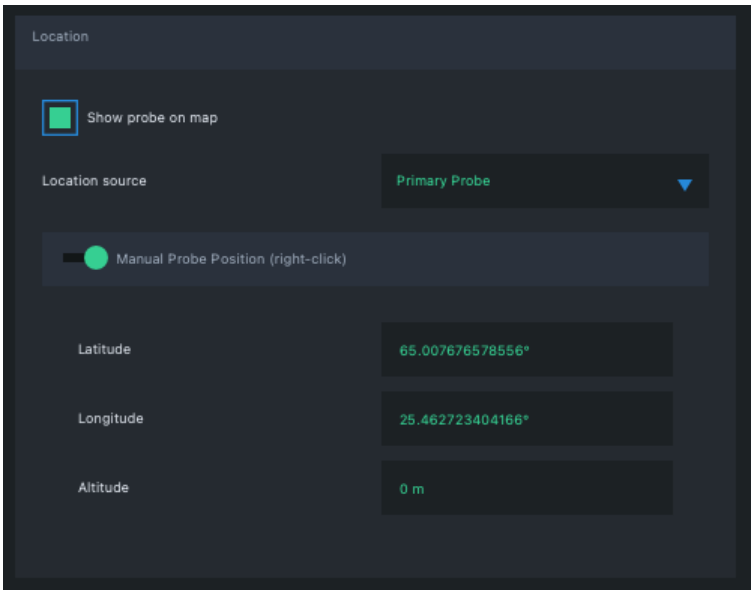
- *3* for GPSD API
- *4* for NMEA messages over UDP. If you want, change the UDP port number for NMEA message reception from the parameter *location_nmea_udp_port*.

After modifying the ini file, restart Qosium Probe.

7.2. Manual Position

If you desire, you can also set the location manually, even repeatedly. For example, if you are doing indoor mobile measurements but don't have an indoor positioning system available, manual positioning can do the trick. Give manual position in **Qosium Scope** as follows:

- On the *Map* leaf, activate the **Manual Probe Position**.
- Select **Location source** according the Probe, Primary or Secondary, you wish to set the location manually.
- Start the measurement.
- Give Probe the position by clicking the right button of your mouse on the map. The position will be shown in the map visualization and stored in the results.



8. Glossary

Network Address Translation

A technique for remapping an IP address space

Wikipedia article on Network Address Translation

QoS Measurement Control Protocol

Kaitotek's proprietary protocol for controlling measurements and gathering measurement results.

QMCP is a protocol made by Kaitotek to optimize QoS measurement control communications. TCP is used in the transport layer (currently), but QMCP controls its sessions. All Qosium products use QMCP.

Network Address Translation

A technique for remapping an IP address space

Wikipedia article on Network Address Translation

Sent Information Not Found

Tämä on erikoistilasto, joka tarkoittaa, että paketti on vastaanotettu, mutta sitä ei ole koskaan merkitty lähetetyksi toisesta päästä.

Kyseessä on siis tavallaan *negatiivinen pakettihäviö*, joka on jo käsitteenä lähestulkoon järjetön. Mikäli SINF-tilastoa esiintyy jatkuvasti, kyseessä onkin usein virheellinen parametrisointi. Jos esimerkiksi mittaussuodatin on jäänyt liian väljäksi, voi mittauspisteelle saapua liikennettä paikasta, jota toisen pään mittauspiste ei ole koskaan nähnyt. Tämä tilasto voi kasvaa ajoittain myös silloin, kun QMCP-yhteys Probejen välillä on niin heikkolaatuinen, että tilannetiedon vaihtaminen ei pysy mittauksen tahdissa. SINF-arvoja voi kuitenkin esiintyä myös normaalissa mittaustilanteessa esimerkiksi silloin, kun mitattava liikenne jotenkin muuttuu matkalla. Hyvä esimerkki tästä on verkkopolun varrella tapahtuva videon muuntokoodaus, jolloin lähettävän puolen mittauspiste näkee pakettinsa hävinneiksi, kun taas vastaanottava pää näkee ilmestyneitä tuntemattomia paketteja.

Virtual Private Network

A technique to provide a secure tunnel over a public network.

Multiprotocol Label Switching

A routing technique in telecommunications networks that forwards data based on short path labels rather than long network addresses.

GPRS Tunnelling Protocol

GTP on nimitys ryhmälle tiedonsiirtoprotokollia, joita käytetään GSM-, UMTS-, LTE- ja 5G-verkkojen sisällä.

Virtual LAN

A LAN seqment which has no physical hardware, but instead is carried over network.

Mean Opinion Score

The most widely used measure for QoE.

Pseudo-Subjective Quality Assessment

A neural network based model for estimating QoE.

For more information, see our article on Quality of Experience.

Generic QoS Measure Algorithm

A parameter based QoS mapping algorithm allowing to map a single quality indicator from several parameters. When tuned with real user tests, GQoSM allows also QoE estimations.

GQoSM, however, is meant for evaluating the influence of the network to the quality – not for estimating the absolute quality (e.g., including the defects of codecs, etc.). For more information, see our article on Quality of Experience.

Round-Trip Time

In packet data communications, RTT is the time it takes a packet to be sent from one network point to another and back.

In practical measurement solutions, like Ping, RTT includes the one-way delays of the directional network paths plus the processing delay of the solution. The processing delay is typically considered small or insignificant compared to the delay caused by the network paths. Sometimes RTT is called Round-Trip Delay, RTD.

Network Time Protocol

A very common protocol for synchronizing the clocks of devices across a network.

Precision Time Protocol

A protocol for synchronizing the clocks of devices across a network. The reached synchronization accuracy is typically considerably better than with NTP.

Global Navigation Satellite System

A general term under which all the different global satellite navigation systems (e.g., GPS, GLONASS, Galileo, BDS) fall.

Pulse per second

A square-like electrical signal that is used in accurate clock synchronization