

QoS Eigenschaften von MQTT

...

Lisa Stolz - Sandra Schuhmacher

Versuchsaufbau

Python Paho

DDR-WRT

Eclipse Mosquitto



client1.py



client2.py



/mqtt-roundtrip-qosX-Y-KB-Zminutes



/mqtt-roundtrip-qosX-Y-KB-Zminutes_2

/mqtt-roundtrip-qosX-Y-KB-Zminutes_2

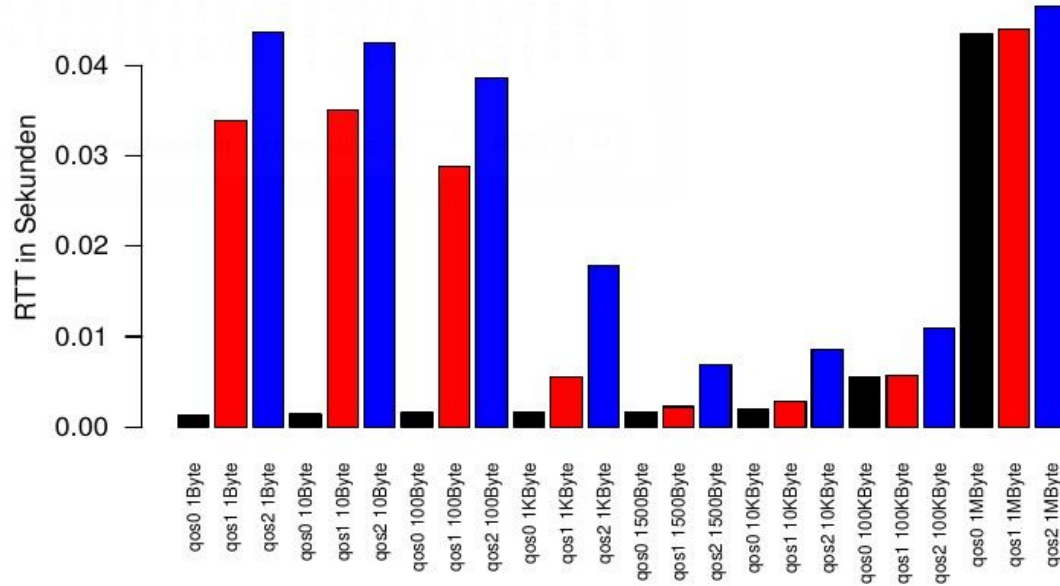
/mqtt-roundtrip-qosX-Y-KB-Zminutes

mosquitto



RTT Messungen bei voller Bandbreite

RTT nach QoS Level und Paketgröße



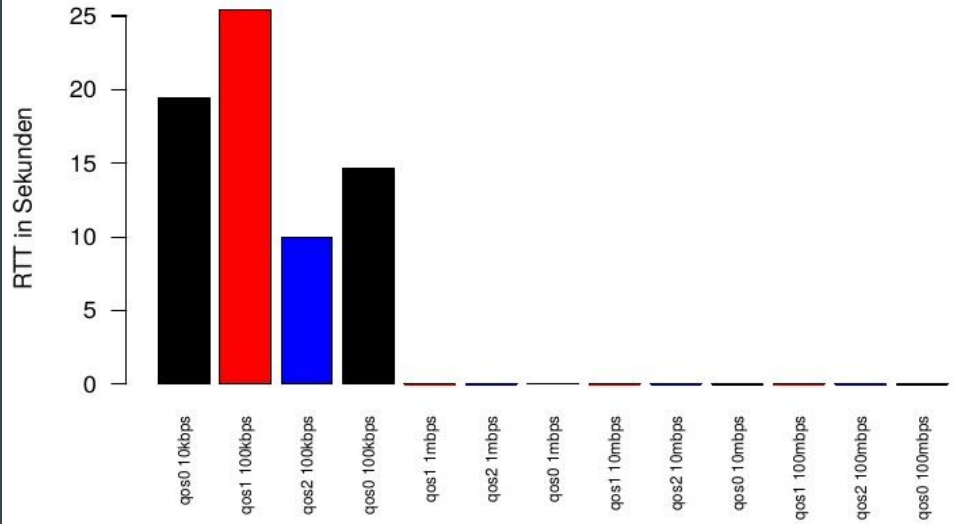
Messgröße	Einheit
Bandbreite	voll
QoS Level	0 , 1, 2
Payload	1 Byte - 1 MByte
Senderate	1, 10, 100 Pakete/Sekunde

- RTT bewegt sich im Millisekundenbereich
- RTT steigt für selbe Payload mit QoS Level
- Ab 1KB Payload sinkt die RTT für QoS1 und QoS2
- Bis 100Byte ist QoS0 konstant und schnell
- Ab 1MB steigt die RTT und QoS0=QoS1=QoS2

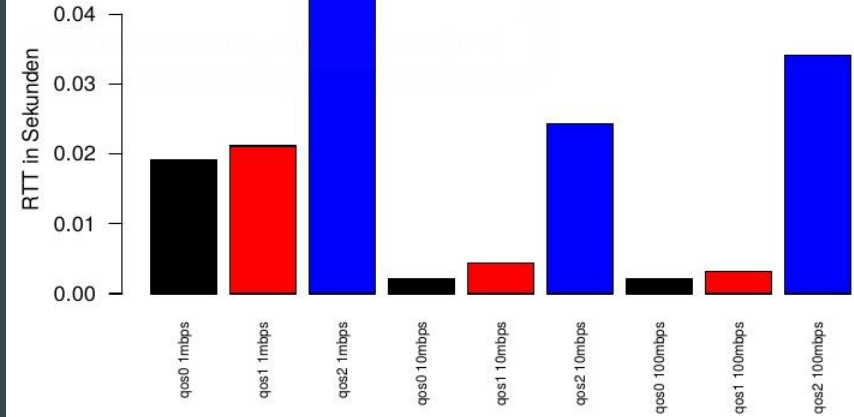
Traffic-Limitierung durch TC

- Bei einer Limitierung auf 10KB pro Sekunde erhöht sich die RTT für alle QoS Level.
- Lässt man 1MB und mehr Traffic zu, verringert sich die RTT auf Millisekunden.

RTT nach QoS und Max Traffic (Paketgröße 10KByte)

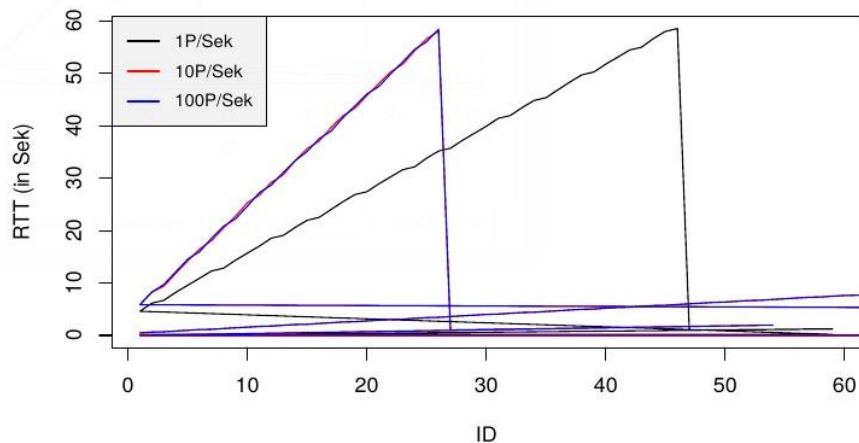


RTT nach QoS und Max Traffic – ohne 10KB und 100KB



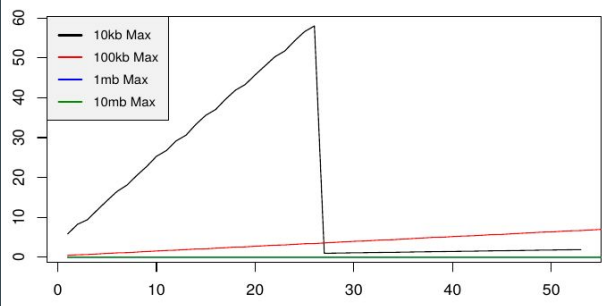
Für genauere Aussagen, müssen die verdichteten Daten aufgefächert werden.

Aufsplittung aller Messungen mit QoS_0 nach Paketen/Sek

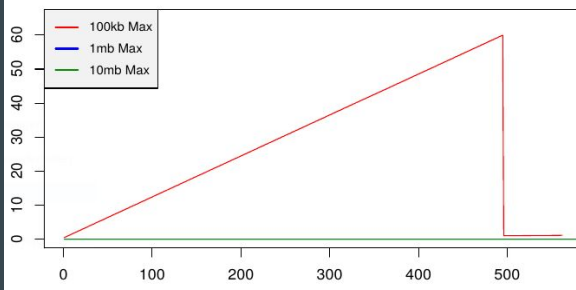


Die “Haifischflosse” wird für verschiedene QoS Level und Begrenzungen beobachtet.

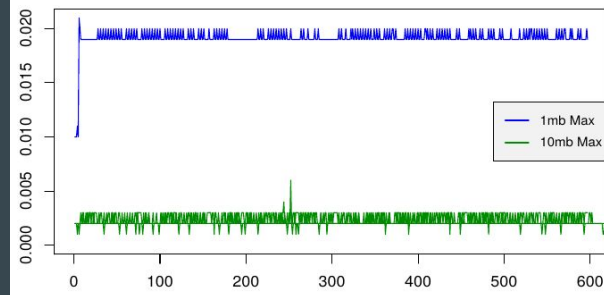
QoS0 10Pakete/Sek aufgeteilt nach Max Durchsatz



QoS0 10Pakete/Sek nach Max (ohne 10kb)



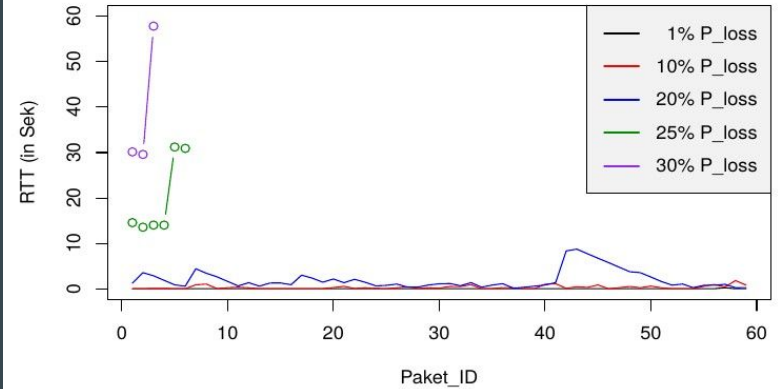
QoS0 10Pakete/Sek nach Max (ohne 10kb und 100kb)



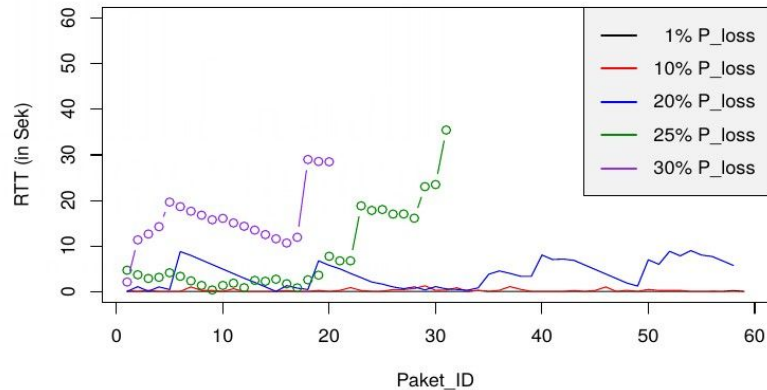
Paketverluste

- Mit der Paketverlustrate steigt die RTT über alle QoS Level.
- Ab 25% Verlustrate bricht die Paketübertragung für jedes QoS Level vorzeitig ab.

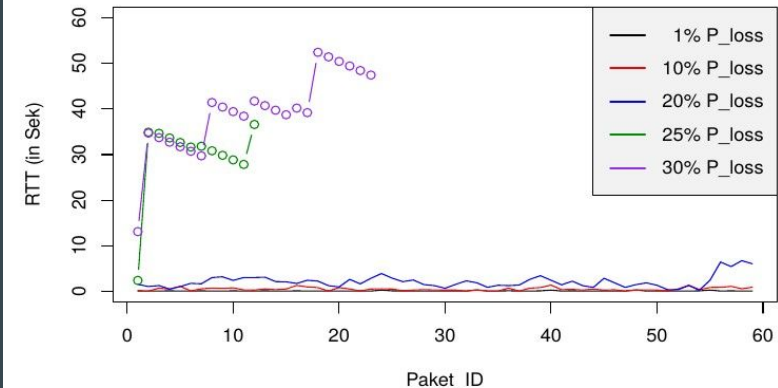
RTT QoS1 (10KByte, 1PproSek)



RTT QoS0 (10KByte, 1PproSek)



RTT QoS2 (10KByte, 1PproSek)



Protokoll Overhead

Netzwerkoverhead für Connection

<i>Message</i>	<i>MQTT</i>	<i>TCP</i>	<i>Pakete</i>
MQTT Connect	25	66	MQTT Connect Command 91 byte
	4	66	MQTT Connect Ack 70 byte
		132	2x TCP ACK je 66 byte
MQTT Disconnect	68	66	MQTT Disconnect Req 68 byte TCP Ack 66 byte

Overhead QoS Level - unabhängig von Payload

<i>QoS Level</i>	<i>MQTT</i>	<i>MQTT+TCP</i>	
QoS 0	0 byte	66 byte	pro Publish nur ein TCP ACK
QoS 1	4 byte	70 byte	Publish Ack im TCP ACK
QoS 2	4 byte	70 byte	Publish Received im TCP ACK
	4 byte	70 byte	Publish Release
	4 byte	70 byte	Publish Complete
	= 12 byte	= 210 byte	

Netzwerkoverhead Status Messages

```
MQ Telemetry Transport Protocol
├── Publish Ack
│   ├── 0100 0000 = Header Flags: 0x40 (Publish Ack)
│   │   ├── 0100 .... = Message Type: Publish Ack (4)
│   │   ├── .... 0... = DUP Flag: Not set
│   │   ├── .... .00. = QoS Level: Fire and Forget (0)
│   │   └── .... ...0 = Retain: Not set
│   ├── Msg Len: 2
│   └── Message Identifier: 10
```

Ethernet 14 bytes	IP 20 bytes	TCP 32 bytes	M Q T T 4
-------------------------	----------------	-----------------	-----------------------

jede Message 4 byte

- QoS1: Publish Ack
- QoS2: Publish Received, Publish Release, Publish Complete

Netzwerkoverhead Publish Message

- Header Flags + Message Länge + Topic Länge + Message
- Ab QoS 1 zusätzlich + Message ID
- 56 byte Overhead im Versuchsaufbau

Header 1B	Msg Len 2B	Msg ID 10B	Topic 43B (max 65535B)	Message (max 256MB)
--------------	---------------	---------------	------------------------------	------------------------

TCP Overhead

- Payload < 1500 byte: Payload mit im MQTT Body geschrieben => kein Overhead
- Payload > 1500 byte: Payload auf mehrere TCP Frames , pro TCP Payload Frame ein TCP ACK
- Transfer Hardware Abhängig - TCP Frames am Anfang nicht voll
 - Einlesegeschwindigkeit der Payload von Festplatte?
 - Netzwerkkarten Buffer?

Message Bundling

Message Bundling - QoS 0

- Publish Bundling bei publishes in kurzen Zeitabständen und kleiner Payload
- Anhängen von Publish an Disconnect Req

14	5.764213711	192.168.1.110	192.168.1.115	MQTT	430 Publish Message, Publish Message, Publish Message,
> Transmission Control Protocol, Src Port: 45870, Dst Port: 1883, Seq: 78, Ack: 5, Len: 364					
v MQ Telemetry Transport Protocol					
v Publish Message					
> 0011 0000 = Header Flags: 0x30 (Publish Message)					
Msg Len: 50					
Topic: mqtt-roundtrip-qos0-1Byte-10-cycles-10pbs					
Message: 000001					
v MQ Telemetry Transport Protocol					
v Publish Message					
> 0011 0000 = Header Flags: 0x30 (Publish Message)					
Msg Len: 50					
Topic: mqtt-roundtrip-qos0-1Byte-10-cycles-10pbs					
Message: 000002					
v MQ Telemetry Transport Protocol					
v Publish Message					
> 0011 0000 = Header Flags: 0x30 (Publish Message)					

Message Bundling - QoS 2

- Kein Publish Bundling - pro Paket ein Publish Received
- Senden mehrerer Publish Release oder Complete Messages in einem Paket oder anhängen von dieser Messages an einen Publish

814	0.097208869	192.168.1.110	192.168.1.115	MQTT	44894 Publish Message, Publish Release, Publish Complete
827	0.099583835	192.168.1.115	192.168.1.110	MQTT	70 Publish Received
828	0.099895609	192.168.1.110	192.168.1.115	MQTT	70 Publish Release
829	0.100143852	192.168.1.115	192.168.1.110	MQTT	78 Publish Complete, Publish Complete

>	Frame 814: 44894 bytes on wire (359152 bits), 44894 bytes captured (359152 bits) on interface 0
>	Ethernet II, Src: WistronI_7d:10:90 (3c:97:0e:7d:10:90), Dst: Giga-Byt_91:59:84 (90:2b:34:91:59:84)
>	Internet Protocol Version 4, Src: 192.168.1.110, Dst: 192.168.1.115
>	Transmission Control Protocol, Src Port: 32886, Dst Port: 1883, Seq: 10441554, Ack: 65, Len: 44828
>	[17 Reassembled TCP Segments (1048632 bytes): #738(26416), #744(65160), #748(65160), #753(65160), #755(65160), #756(65160), #757(65160), #758(65160), #759(65160), #760(65160), #761(65160), #762(65160), #763(65160), #764(65160), #765(65160), #766(65160), #767(65160), #768(65160)]
✓	MQ Telemetry Transport Protocol
>	Publish Message
✓	MQ Telemetry Transport Protocol
>	Publish Release
✓	MQ Telemetry Transport Protocol
>	Publish Release
>	0110 0010 = Header Flags: 0x62 (Publish Release)
>	0110 = Message Type: Publish Release (6)

Danke fürs zuhören!

Neugierig geworden? Selber testen? Die Logs sehen?



<https://github.com/yulivee/mqtt-qos-rountrip>

Lessons Learned

Wie man einen Broker abschießt

- Mosquitto stürzt ab wenn er mit dem persistieren nicht hinterherkommt
 - Log messages
 - Persistent messages (In default config aktiviert)
- Zuviel Messages pro Sekunde
- Langsames Schreibmedium
- Verarbeitungsgeschwindigkeit

Client Library Einschränkungen

- Paho kann auf Paketloss beim Connect seltsam reagieren => erschwert automatische reconnects
 - python friert im `client.connect()` fest
 - Interrupted system call bei Verlust des connect-pakets

MQTT Messages sind anonym

- Ein Paket hat keine eindeutige ID
- Ein Client kann nicht wissen von welchem anderen Client ein Paket kam
- Reihenfolge der Pakete kann nicht festgestellt werden

Backup

Erklärungsansatz Haifisch und niedrige RTT für QoS2:

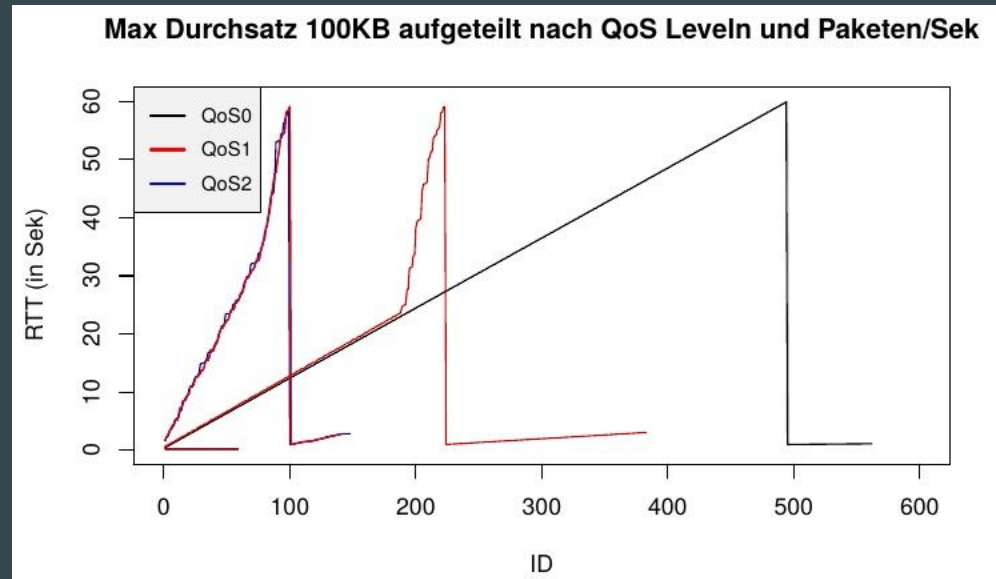
Durch die verringerte Bandbreite scheint es zu einer Art “Stau” zu kommen. Die RTT steigt mit jedem neuen Paket und bricht schließlich wieder ein auf wenige Millisekunden (Haifischflosse).

Im Gegensatz zu QoS0 und QoS1 sendet QoS2 für starke Begrenzungen (100KB) nicht mit 10 oder 100 Paketen pro Sekunde - die Haifischflosse wird seltener aber früher beobachtet!

Weniger Beobachtungen (versandte Pakete) für QoS2 bei höherer Sendungsrate:

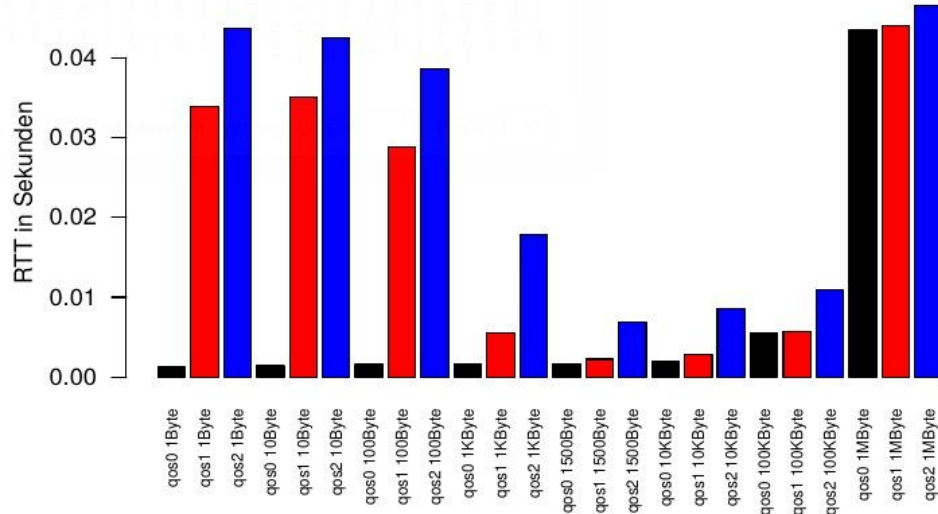
- QoS0: 59(1pbs), 562(10pbs), 562(100pbs)
- QoS1: 59(1pbs), 141(10pbs), 383(100pbs)
- QoS2: 59(1pbs), 141(10pbs), 148(100pbs)

=> geringeren RTT Durchschnittswerten

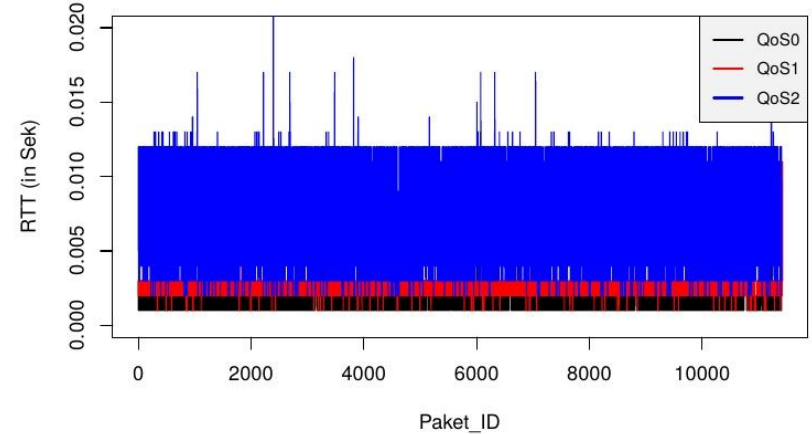


- RTT bewegt sich im Millisekundenbereich
- RTT steigt für selbe Payload mit QoS Level
- Ab 1KB Payload sinkt die RTT für QoS1 und QoS2
- Bis 100Byte ist QoS0 konstant und schnell
- Ab 1MB steigt die RTT und QoS0=QoS1=QoS2

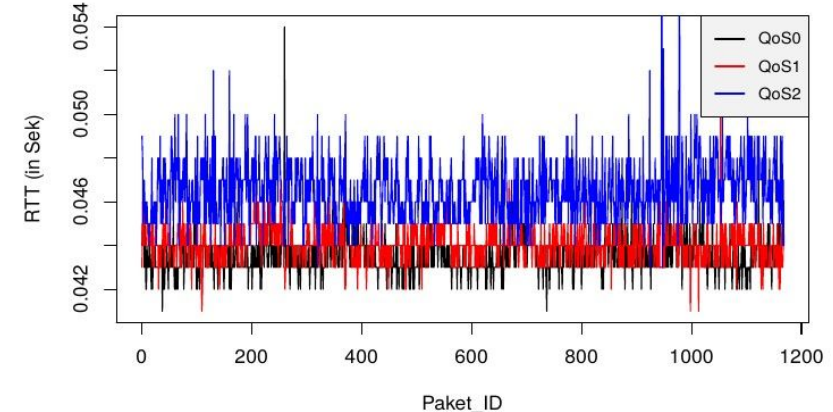
RTT nach QoS Level und Paketgröße



Paketgröße 1500Byte Aufsplittung nach QoS Level

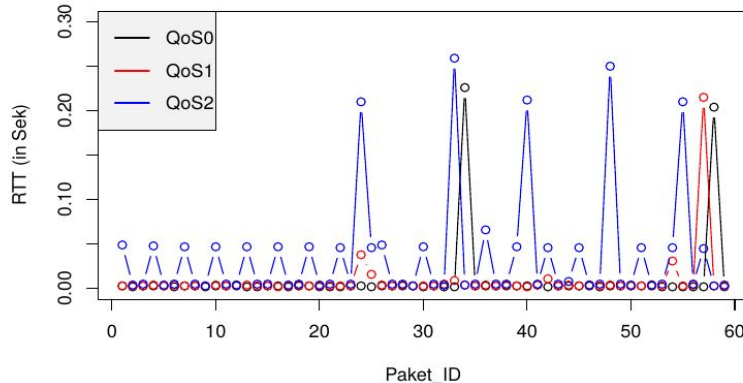


Paketgröße 1MByte Aufsplittung nach QoS Level

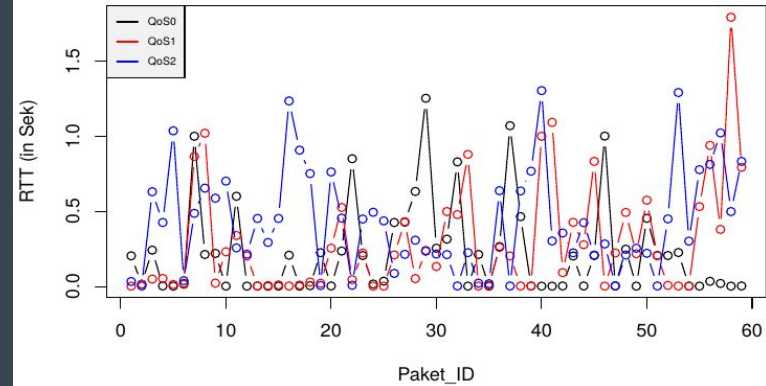


Mit der Paketverlustrate steigt die RTT von wenigen Millisekunden ab 10% Verlustrate auf mehrere Sekunden an.

RTT Paketloss 1% (10KByte, 1PproSek)



RTT Paketloss 10% (10KByte, 1PproSek)



RTT Paketloss 20% (10KByte, 1PproSek)

