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

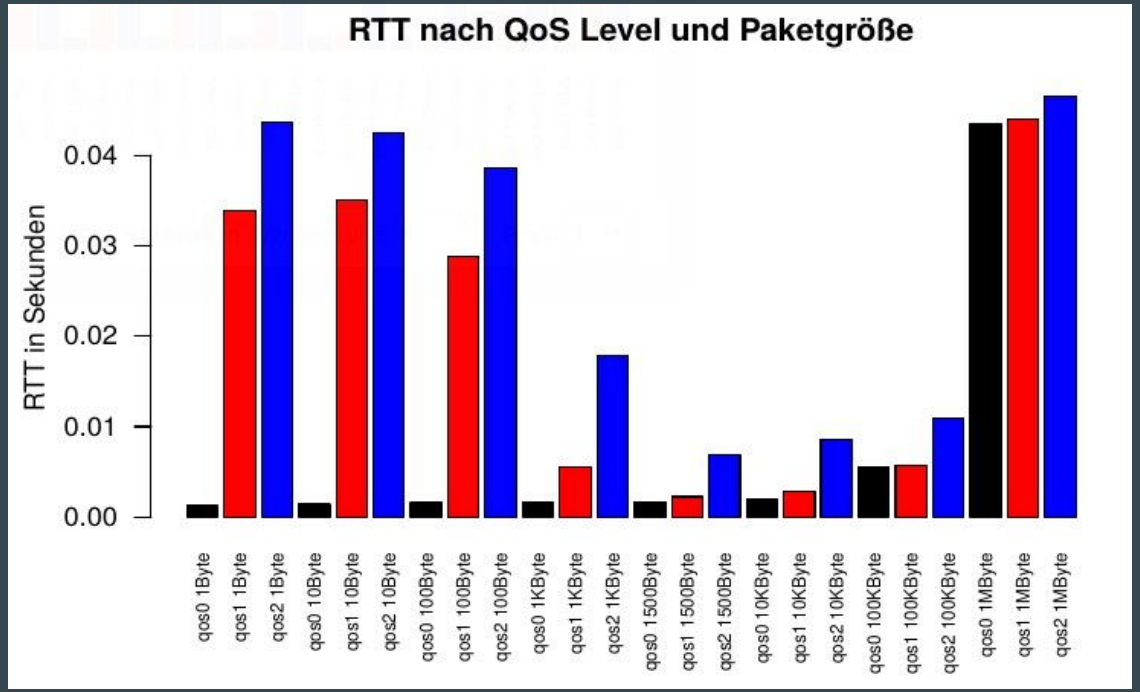
/mqtt-roundtrip-qosX-Y-KB-Zminutes_2

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

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

mosquitto

RTT Messungen bei voller Bandbreite

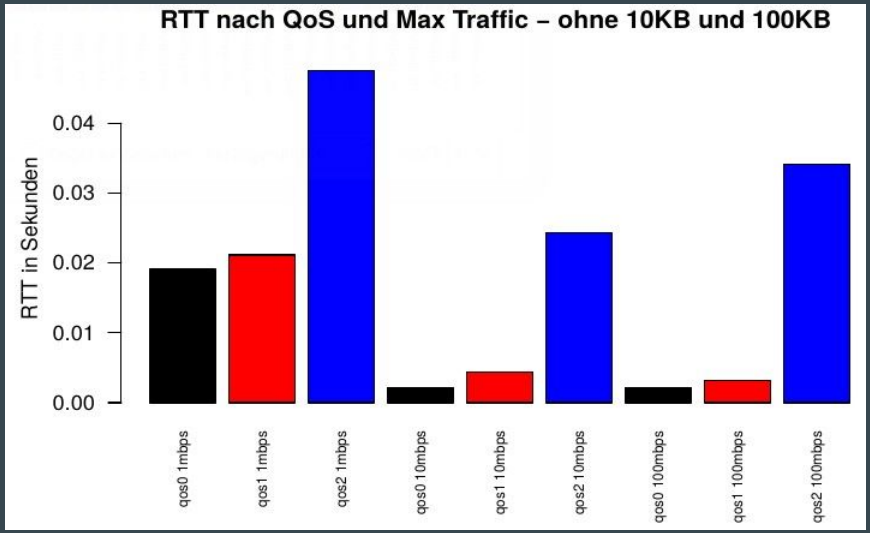


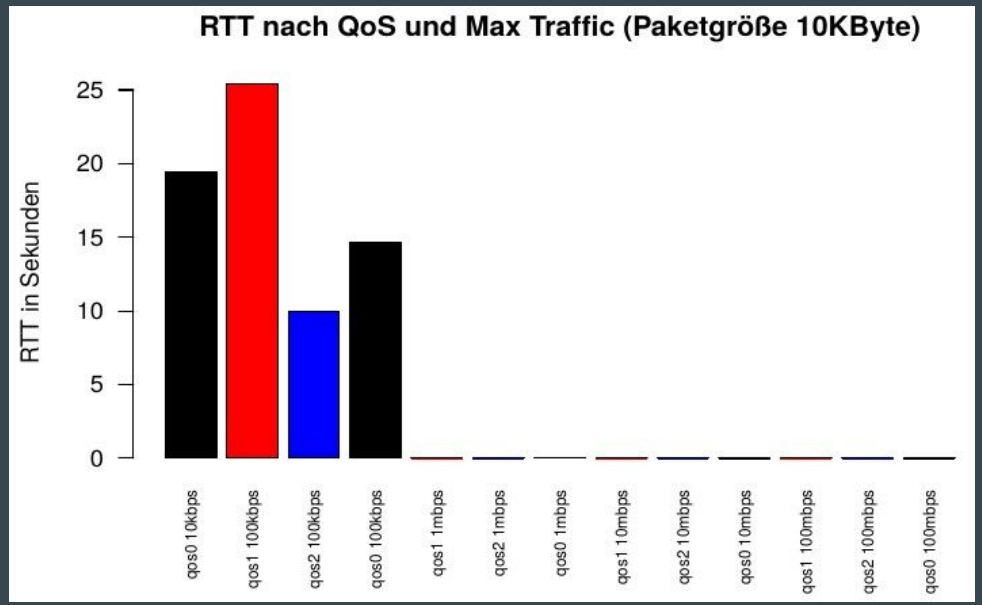
Messgröße	Einheit
Bandbreite	voll
QoS Level	0 , 1, 2
Payload	1 Byte - 1 MByte
Senderate	1, 10, 100 Pakete/Sekunde

- 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

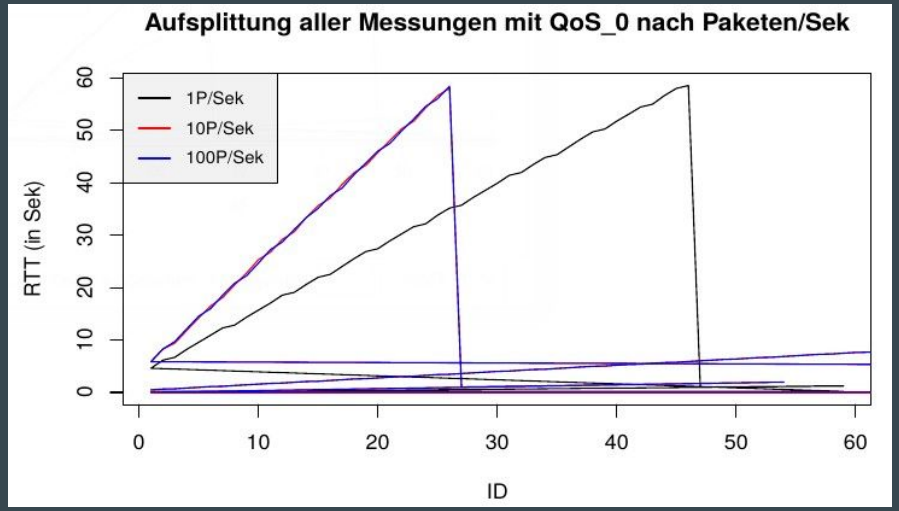
Traffic-Limitierung durch TC

- 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.

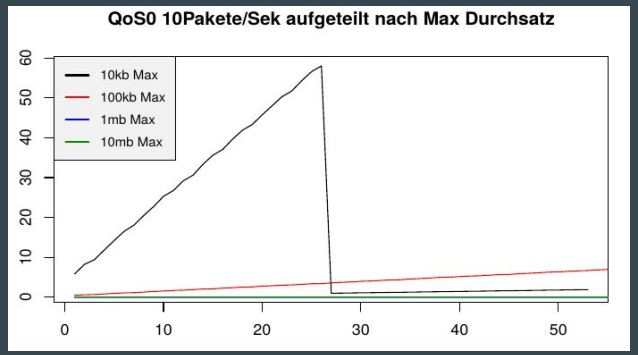


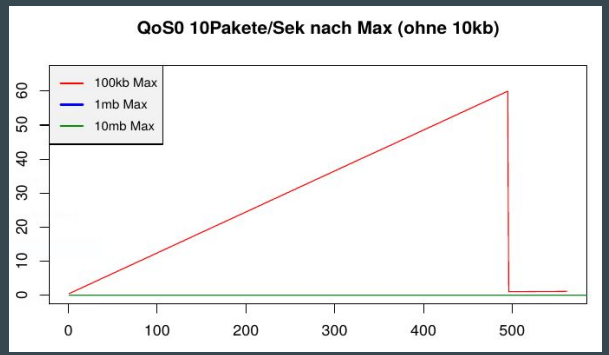


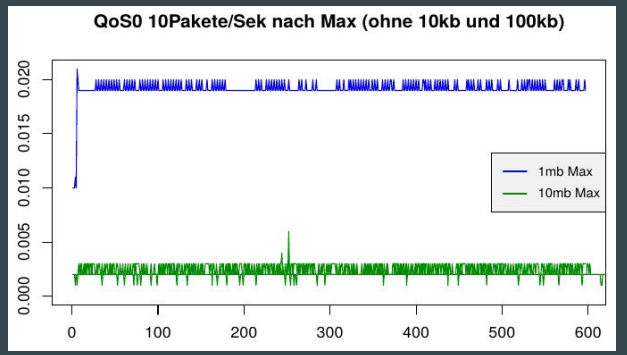
Für genauere Aussagen, müssen die verdichteten Daten aufgefächert werden.



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

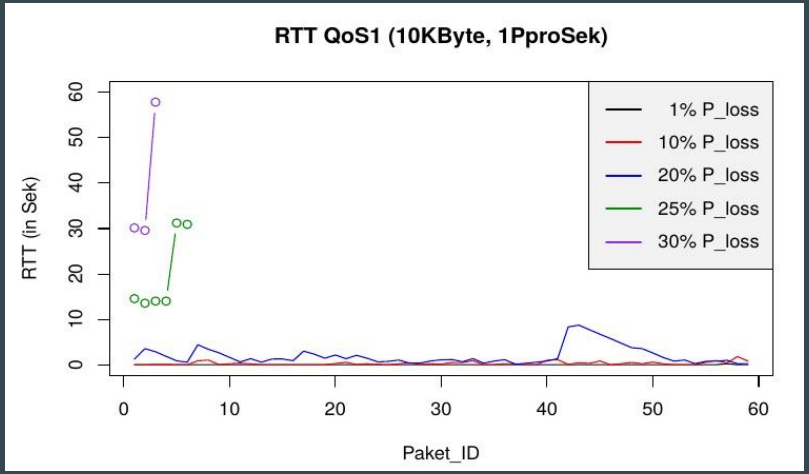


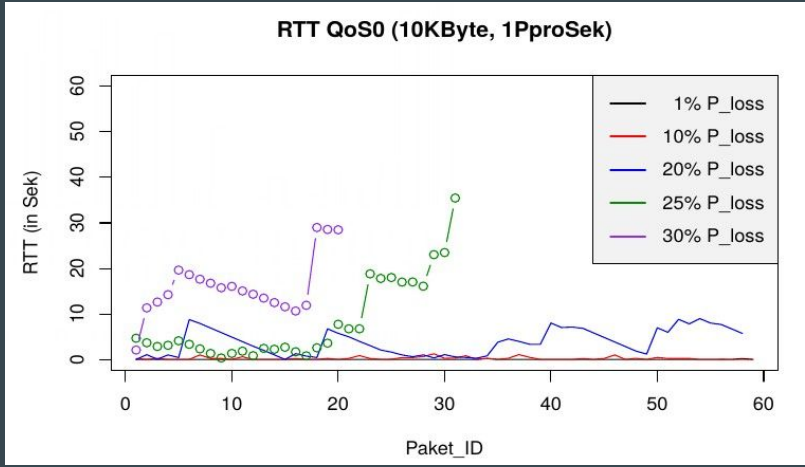


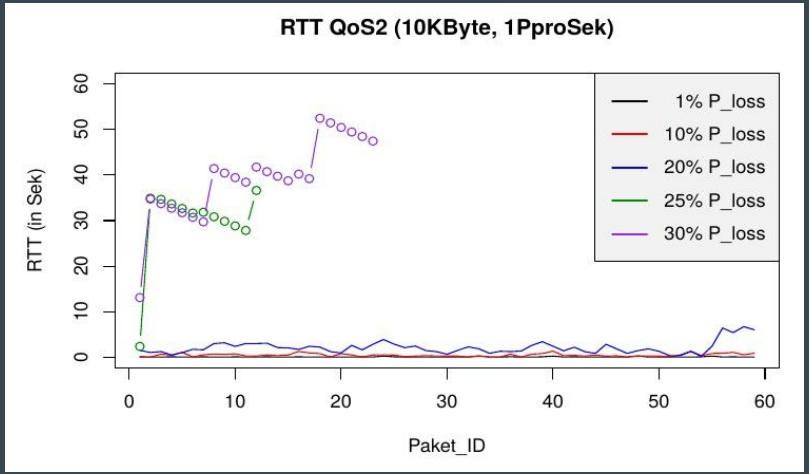


Paketverluste

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







Protokoll Overhead

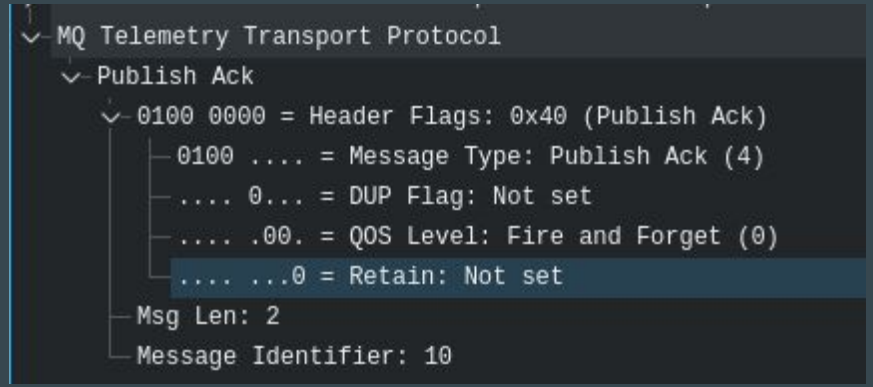
Netzwerkoverhead für Connection

Message	*MQTT*	*TCP*	*Pakete*
MQTT Connect	25 4	66 66 132	MQTT Connect Command 91 byte MQTT Connect Ack 70 byte 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

QoS Level	*MQTT*	*MQTT+TCP*	
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 4 byte 4 byte = 12 byte	70 byte 70 byte 70 byte = 210 byte	Publish Received im TCP ACK Publish Release Publish Complete

Netzwerkoverhead Status Messages



Ethernet 14 bytes	IP 20 bytes	TCP 32 bytes	MQTT 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)

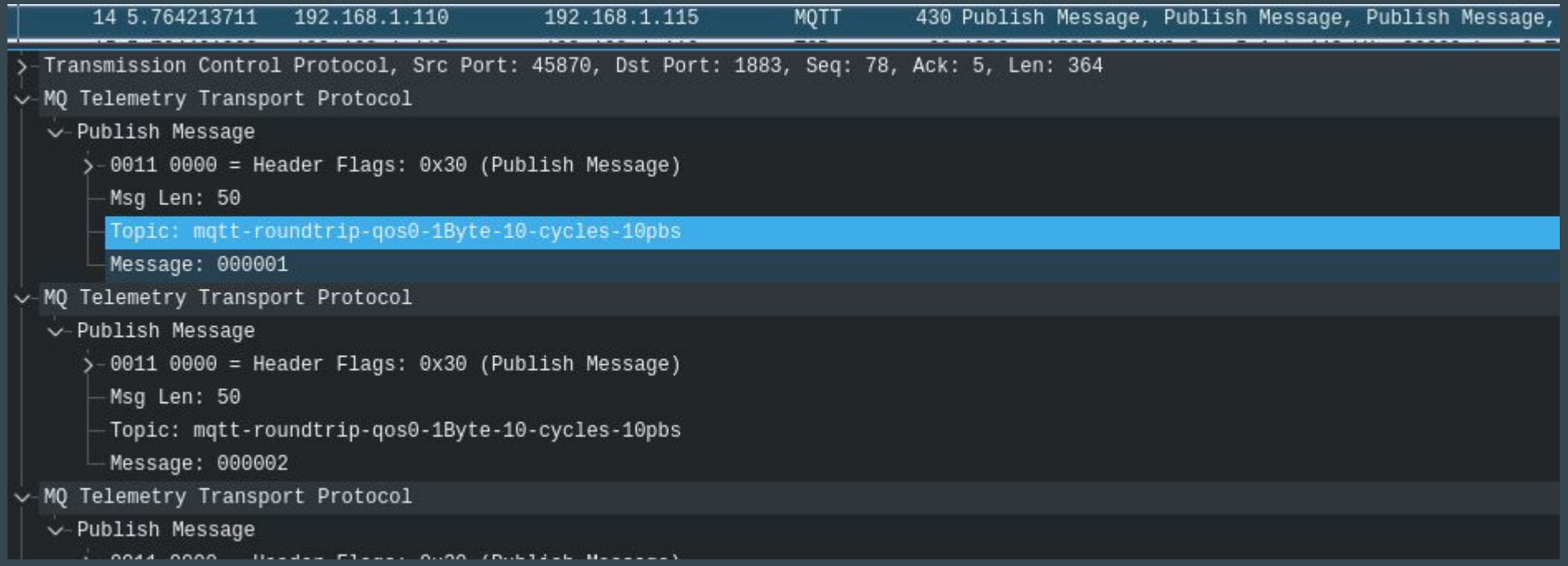
TCP Overhead

- 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

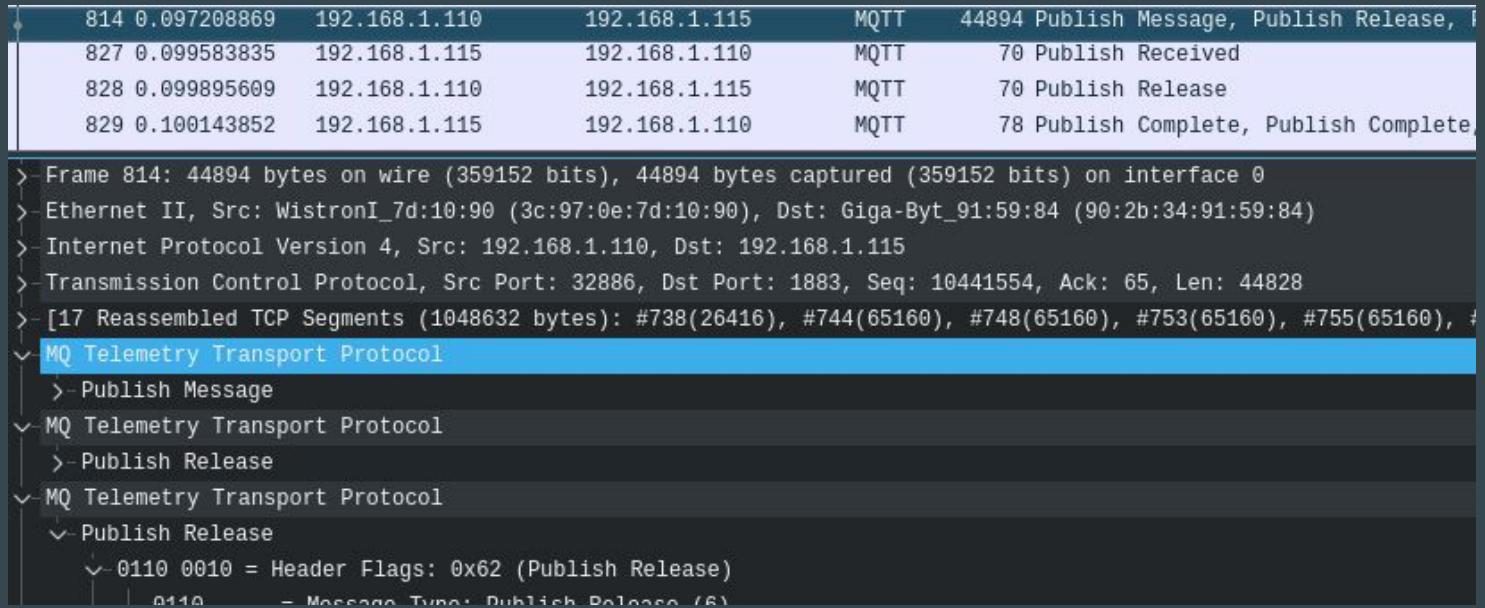
Message Bundling - QoS 0

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



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



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:

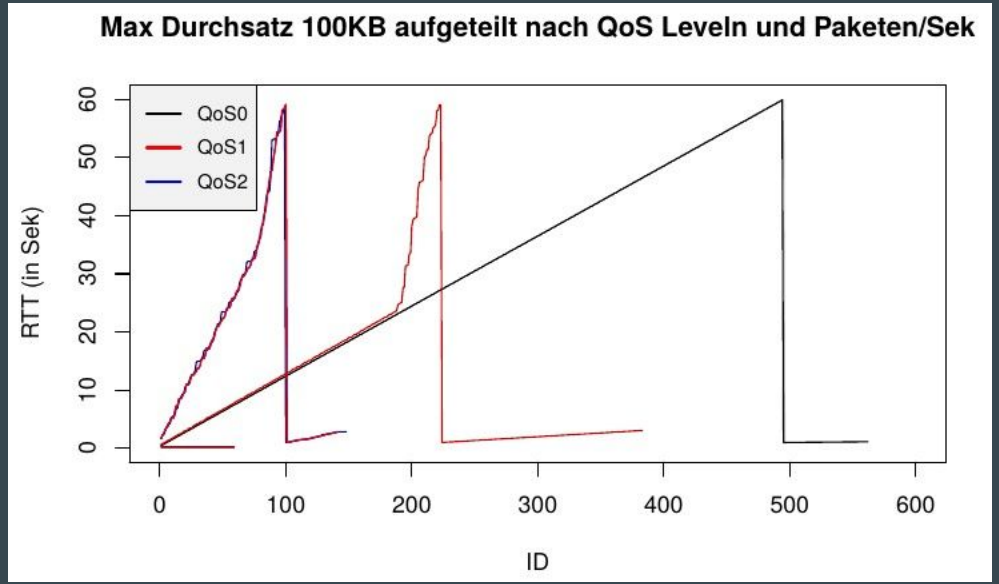
Durch die verringerte Bandbreite scheint es zu einer Art “Stau” zu kommen. Die RTT steigt mit jedem neuen Paket und bricht schließlich wieder ein auf wenige Millisekunden (Haifischflosse).

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

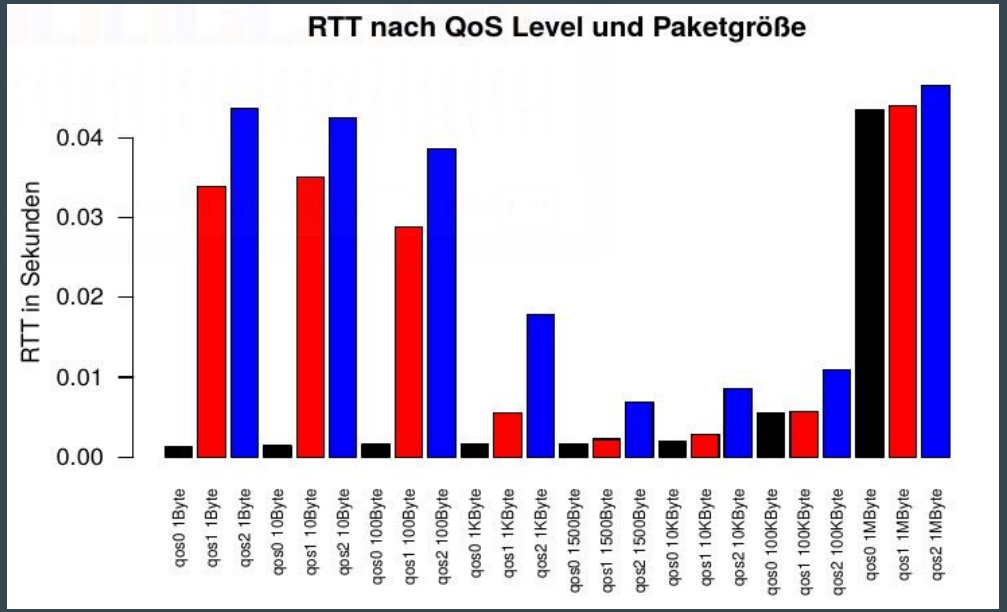
Weniger Beobachtungen (versandte Pakete) für QoS2 bei höherer Sendungsrate:

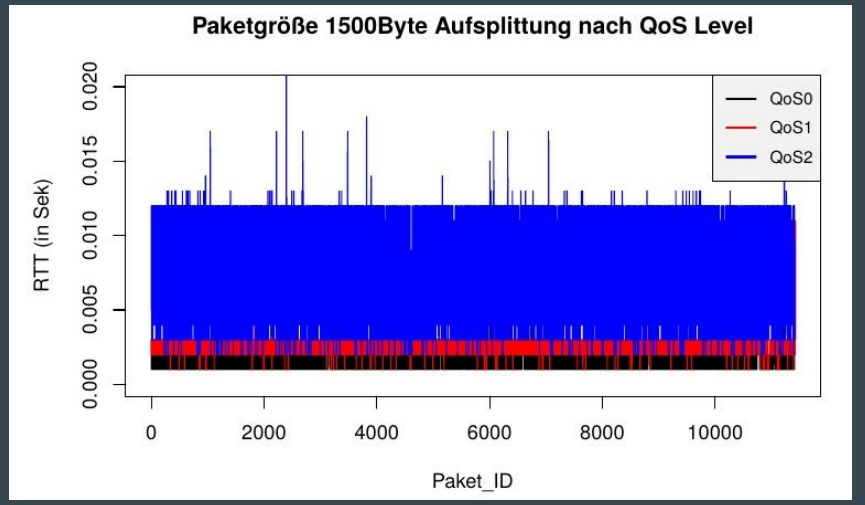
- QoS0: 59(1pbs), 562(10pbs), 562(100pbs)
- QoS1: 59(1pbs), 141(10pbs), 383(100pbs)
- QoS2: 59(1pbs), 141(10pbs), 148(100pbs)

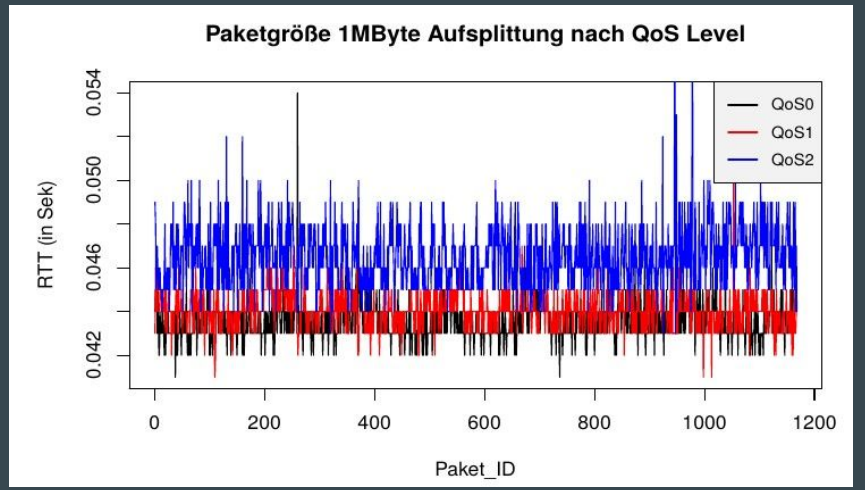
=> geringeren RTT Durchschnittswerten



- 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







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