

MQTT-Benchmark

Patrick Olinger und David Tarnow

Zielsetzung

Implementationen des MQTT-Protokolls, in verschiedenen Sprachen, werden auf ihren Ressourcenbedarf untersucht. Genutzt wird der Paho MQTT-Client in C und C# sowie der Qt MQTT-Client in C++. Als Broker wird Mosquitto genutzt.

Gemessen werden der Arbeitsspeicher und die CPU-Auslastung der jeweiligen Clients als Publisher und Subscriber.

Dabei wird einerseits die Auswirkung der Nachrichtengröße und andererseits die Auswirkung des Publishing-Intervalls ausgewertet.

Zum Auswerten der Nachrichtengröße senden die Publisher in 100 ms Intervallen Nachrichten verschiedener Größen. Die Datenmengen betragen entweder 0, 1.000, 100.000 oder 1.000.000 Bytes.

Zum Auswerten des Einflusses des Zeitintervalls wird eine feste Nachrichtengröße von 100.000 Bytes genutzt und die Zeitintervalle der Publisher auf 10, 100 und 1.000 ms gesetzt.

Implementation

Die MQTT-Clients laufen bei jeder Implementation unter Windows. Der Mosquitto Broker läuft auf einer virtuellen Maschine unter Linux.

Die Publisher veröffentlichen in gegebenen Zeitintervallen die Daten unter einem bestimmten Topic.

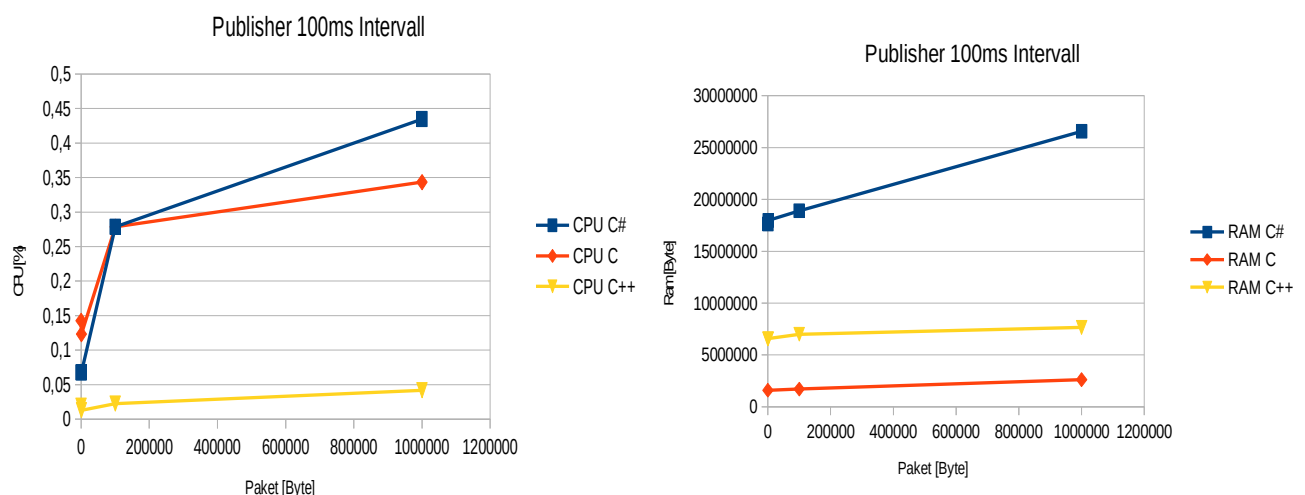
Die Subscriber beobachten dieses Topic um neue Daten zugesendet zu bekommen.

Zum Erheben der Messdaten wird ein eigens in C++ geschriebenes Programm verwendet, welches jede Sekunde die Auslastung des Arbeitsspeichers sowie der CPU ausliest.

Auswirkung der Nachrichtengröße

Bei dieser Messreihe werden Nachrichten verschiedener Größe bei einem Intervall von 100 ms veröffentlicht. Die Nachrichtengröße beträgt 0, 1.000, 100.000 und 1.000.000 Bytes.

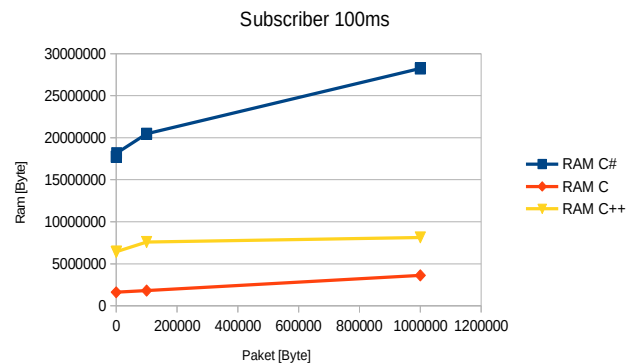
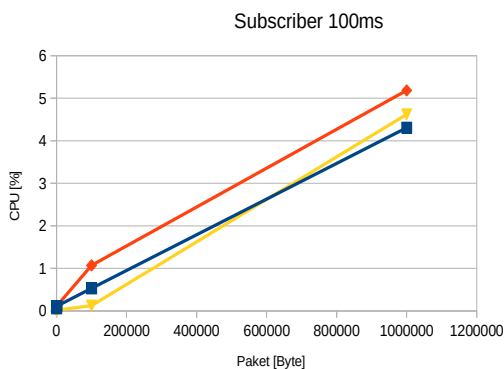
Für die Auswertung wurde zwei Minuten lang die CPU und RAM Auslastung gemessen und jeweils der Mittelwert gebildet.



Auf den Abbildungen der Publisher ist zu erkennen, dass bei C# die größte RAM Auslastung ist, die mit steigender Nachrichtengröße auch weiter wächst. Während die CPU Auslastung bei Paketen ohne Payload niedriger ist als die der C Implementation jedoch bei steigender Größe stark ansteigt. Bei den C und C++ Implementationen ist kein großer Anstieg des Arbeitsspeichers zu erkennen. C++ verzeichnet auch kaum eine erhöhte CPU Auslastung. C dagegen hat eine höhere CPU Auslastung bei größeren Paketen.

Der ähnliche Anstieg im CPU Bedarf bei der C und C# Implementation kann darauf zurückgeführt werden, dass in beiden Fällen die Paho MQTT Library genutzt wird.

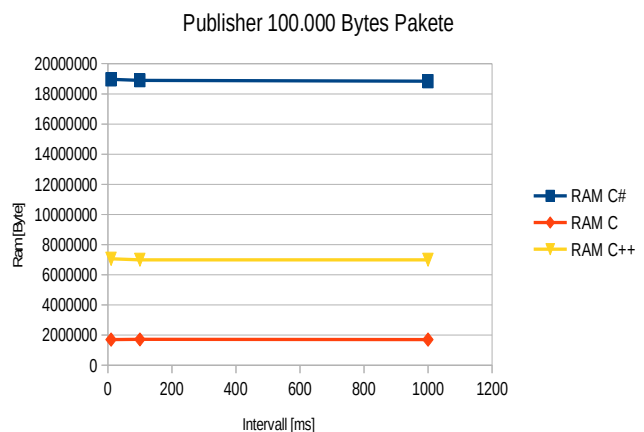
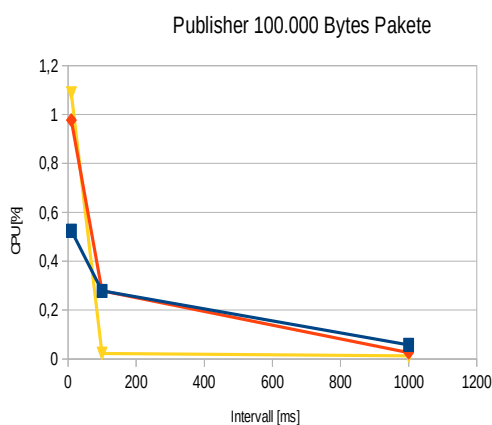
Die hohe RAM Auslastung bei C# lässt sich durch den durch C# erzeugten Overhead erklären.



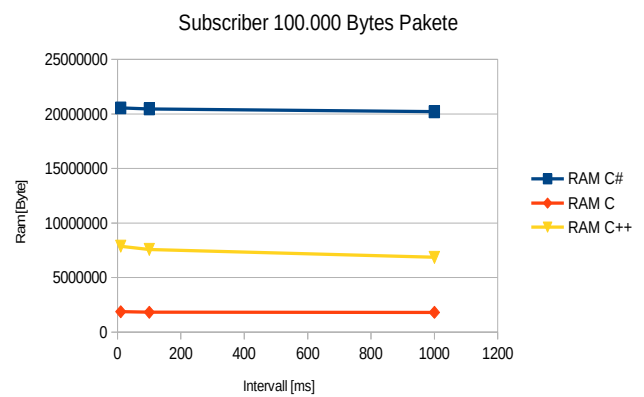
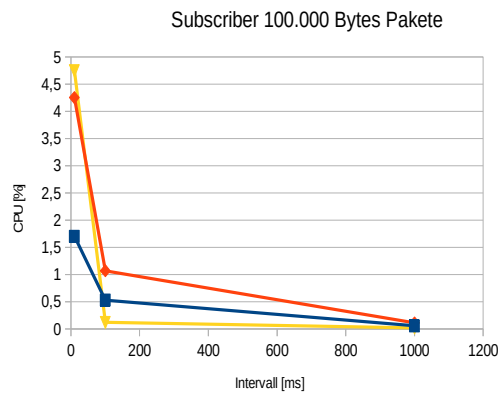
Auch bei den Subscribern ist ein erhöhter RAM Bedarf beider C# Implementation zu sehen. Die CPU Auslastung steigt hier jedoch bei allen drei Implementationen stark an mit steigender Paketgröße. Der Unterschied zu den Publishern ist in diesem Fall, dass die Publisher bei Programmstart initial den Speicher für die Nachrichten allokatieren und bei den Subscribern bei jeder empfangenen Nachricht neu.

Auswirkungen des Publishingintervalls

Als nächstes wird die Auswirkung des Publishingintervalls bei fester Datenmenge gemessen. Dazu wird eine Nachrichtengröße von 100.000 Bytes in den Intervallen 10, 100 und 1000 ms gesendet. Es wird jeweils der Durchschnitt aus zwei Minuten gewertet.



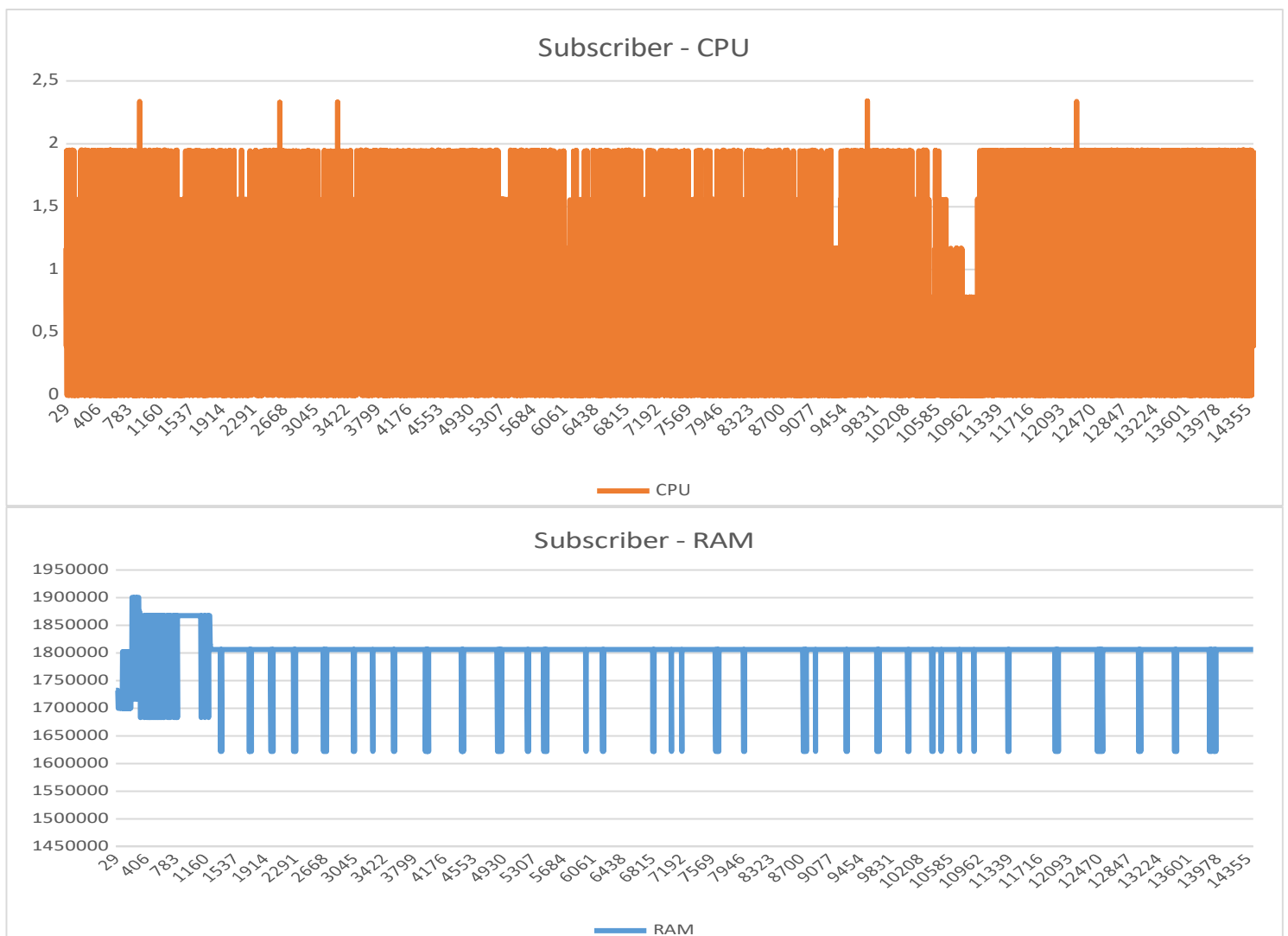
Bei steigender Intervallgröße sinkt die CPU Auslastung ab. Hier ist zu erkennen, dass die Auslastung der Paho Implementationen im gleichen Maße sinkt. Bei gleicher Paketgröße ist bei allen Implementationen kein Anstieg des Arbeitsspeicherverbrauchs zu erkennen.

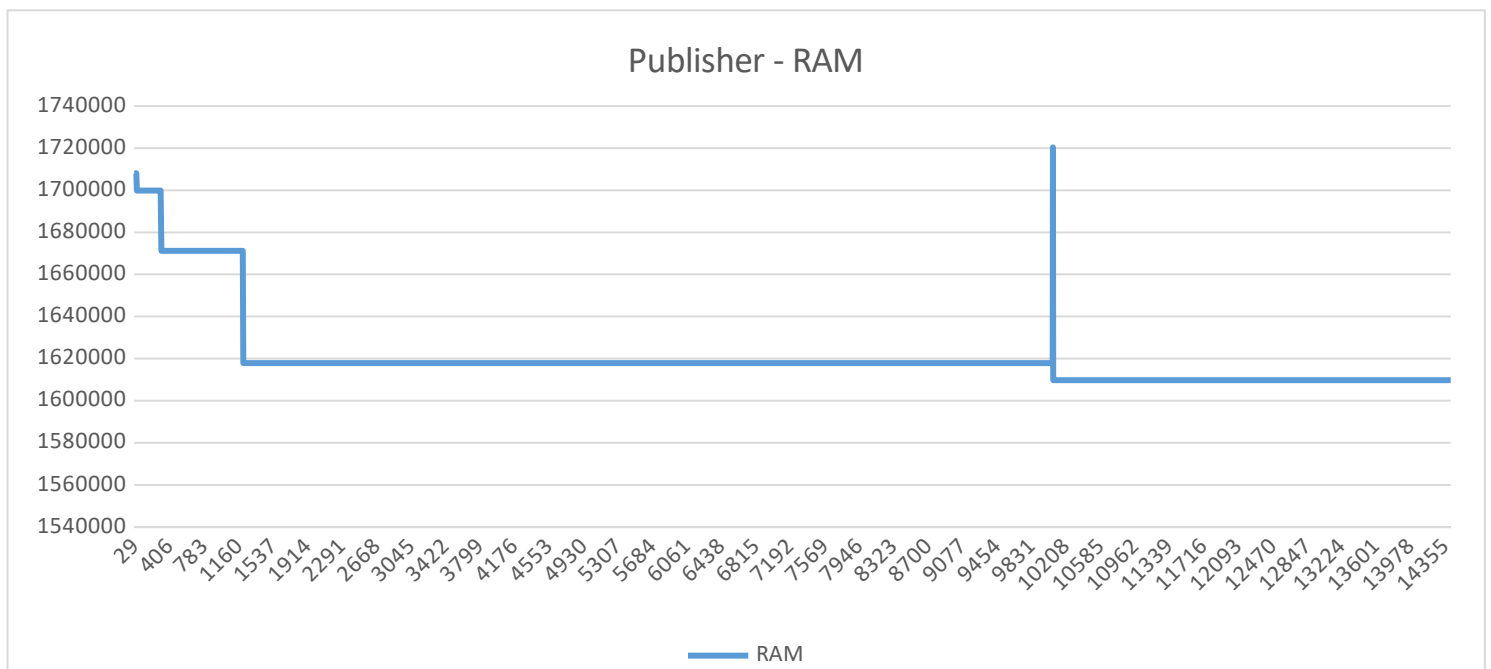
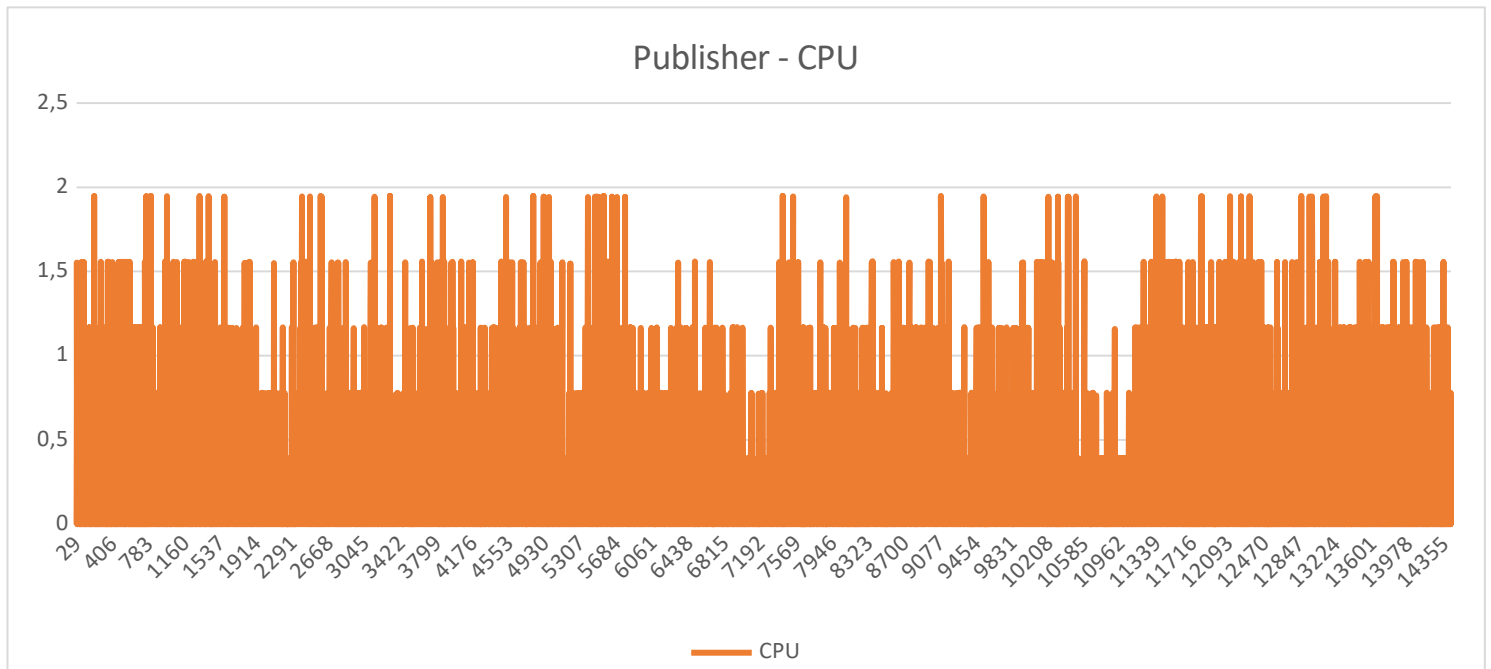


Auch bei den Subscribern ist eine sinkende CPU Auslastung bei höheren Frequenzen zu sehen. Die RAM Auslastung ist bei allen Implementationen bei allen Intervallen nahezu gleichbleibend.

Langzeitmessungen

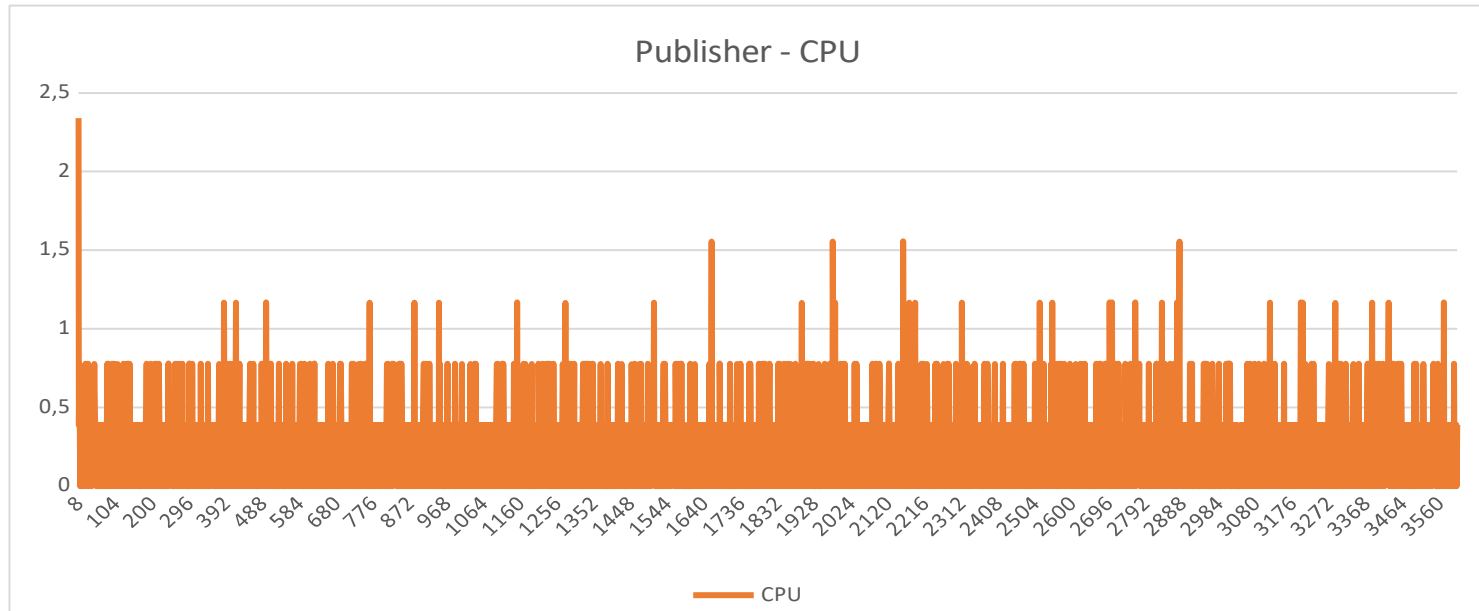
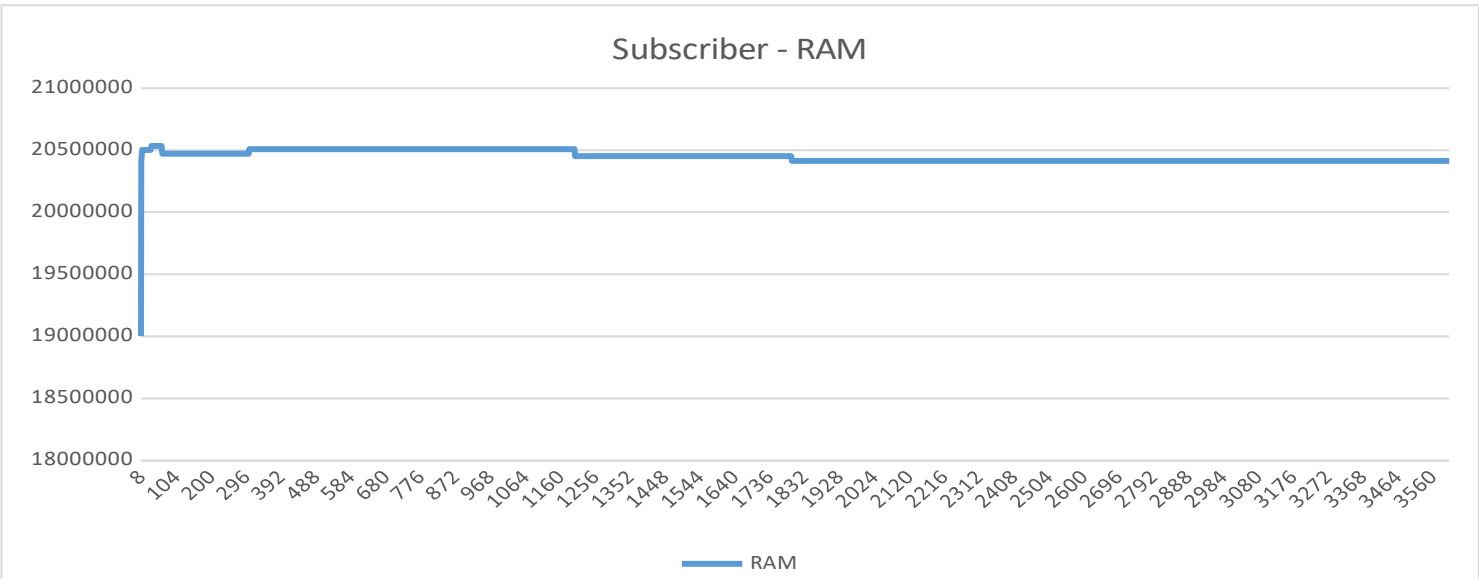
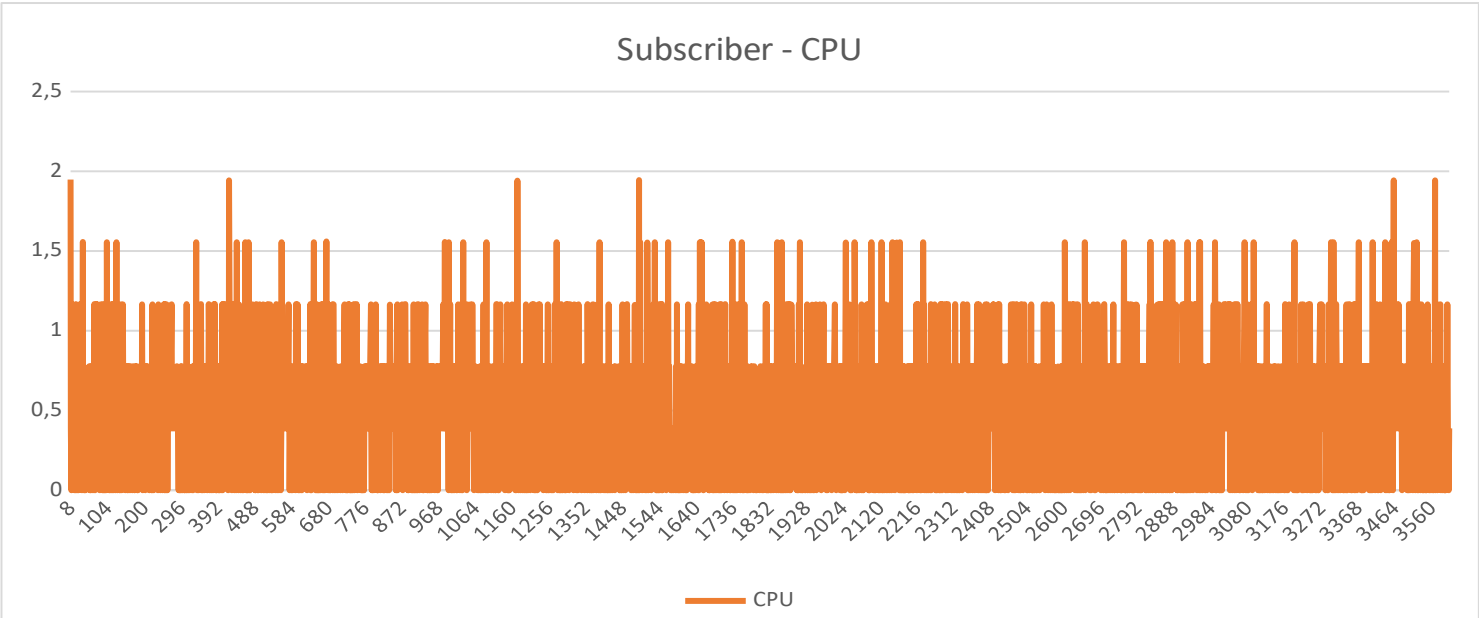
C:



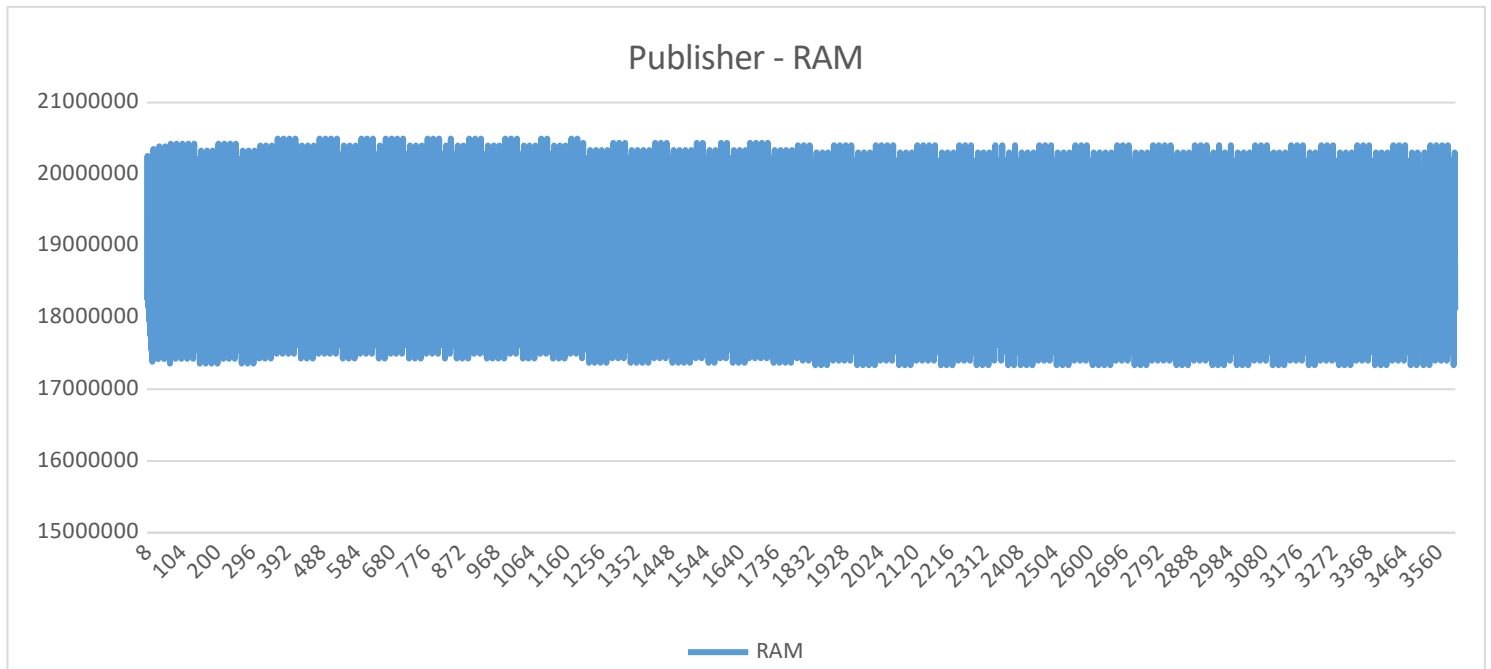


Bei den Langzeitmessungen in C ist zu erkennen, dass sich der RAM und CPU Bedarf sehr stabil verhält bis auf ein paar Ausreißer.

C#:

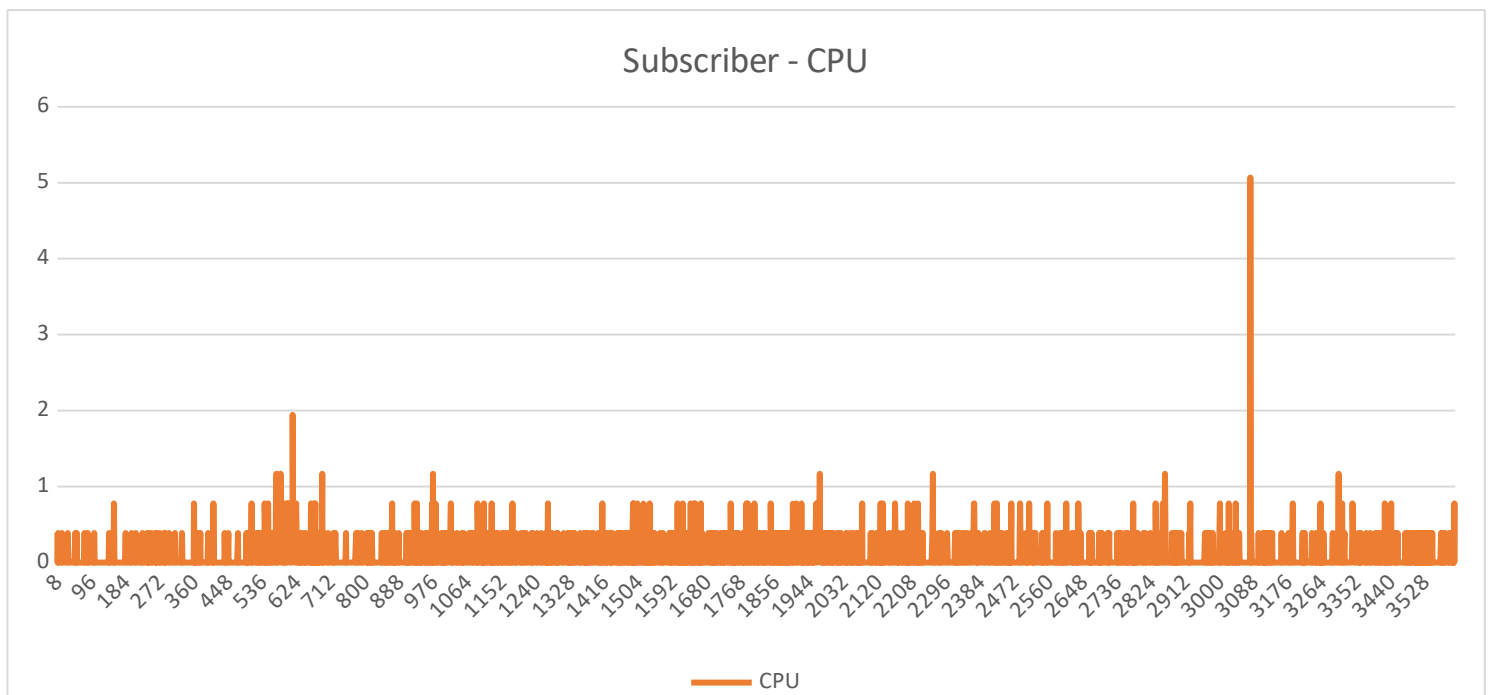


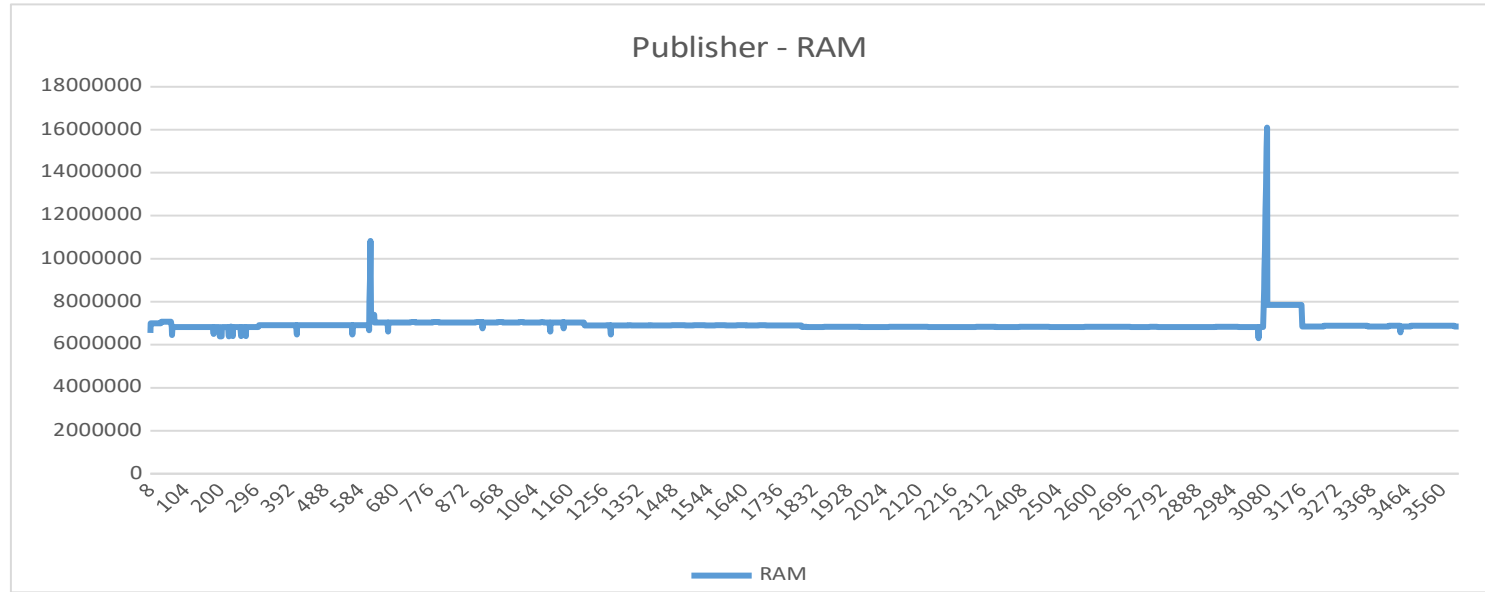
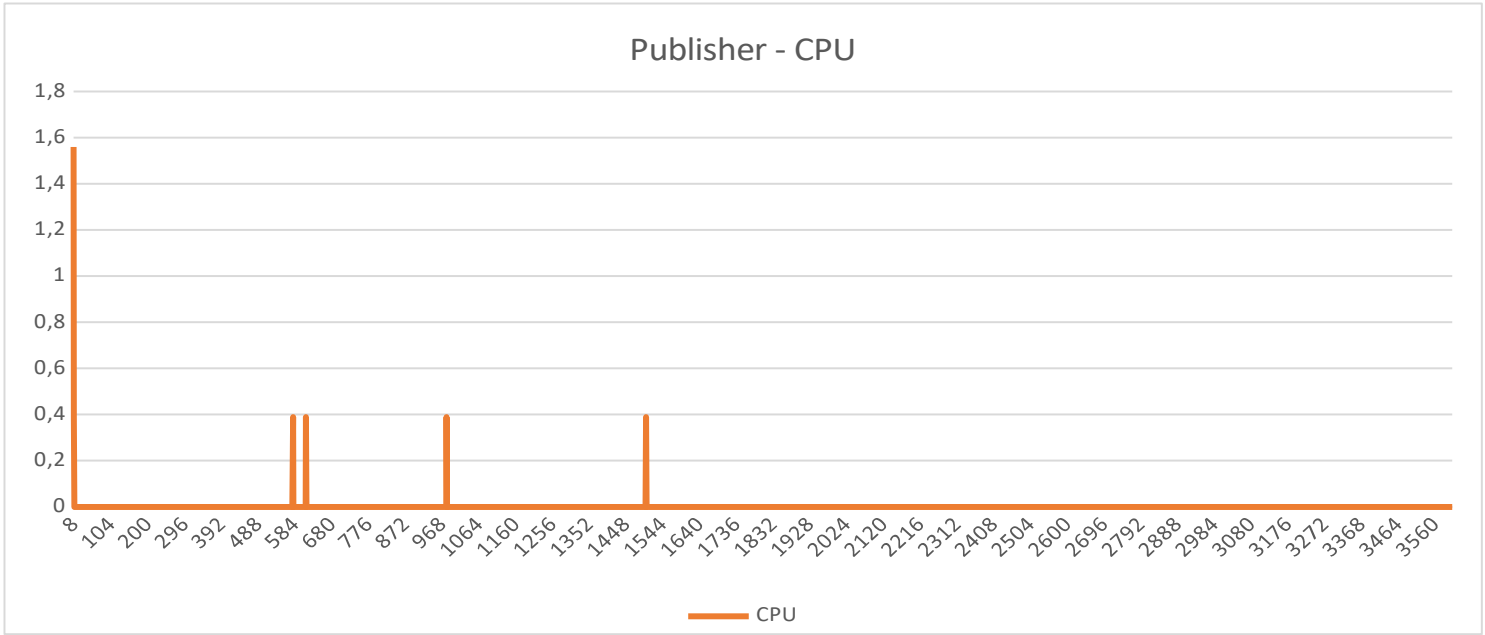
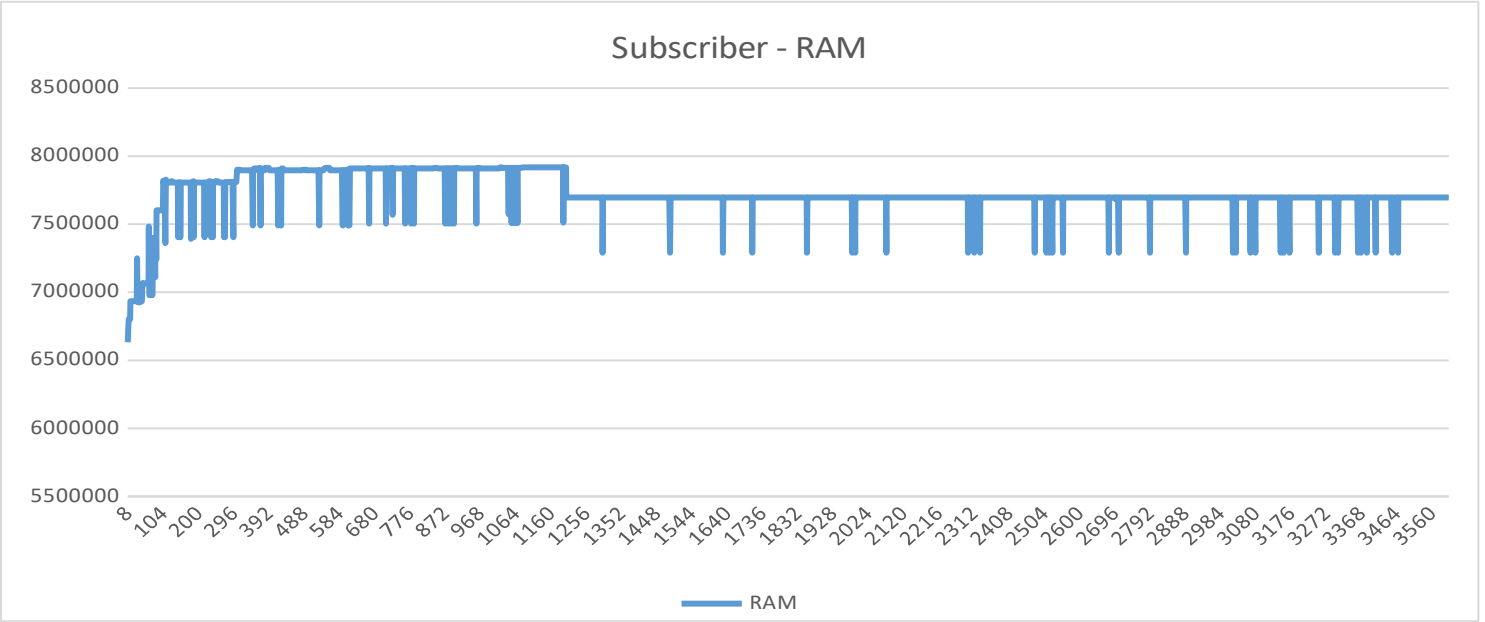
u



Auch bei der C# Implementation ist kein dauerhafter Anstieg an CPU oder Arbeitsspeicherbedarf zu vermerken.

C++:





Auch die C++ Langzeitmessungen zeigen außer ein paar Auffälligkeiten wie erwartet ähnliches Verhalten wie die Kurzzeitmessungen.

Fazit

In Bezug auf Arbeitsspeicherverbrauch ist die C Implementation vorne. Bei allen Messungen hat diese den niedrigsten Verbrauch, dicht gefolgt von der C++ Implementation. C# hat eine viel größere RAM Auslastung.

Beim CPU-Bedarf liegt die C++ Implementation vorne, ist aber in den meisten Fällen kein nennenswerter Unterschied zu den anderen beiden Implementationen. Die C und C# Implementationen haben meist eine ähnliche Auslastung die bei größeren Nachrichten oder kleinen Intervallen weiter auseinander geht, wobei C meist weniger Ressourcen verwendet.

Zeitschätzung

Einarbeitung Dokumentation:

C#: 30 min

C: 30 min

C++: 30 min

Erstes Beispiel:

1h

Erstellung der Demo:

15h

Erstellung der Dokumentation:

8h (zusammen mit Auswertung der Messdaten)