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 Clienten 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.

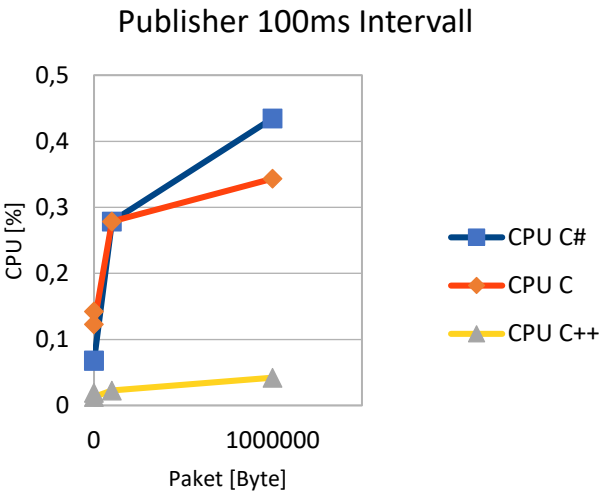
Implementation

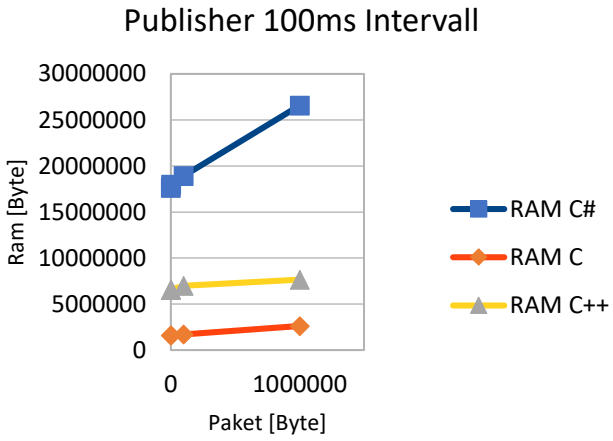
Die MQTT-Clients laufen bei jeder Implementation unter Windows. Der Mosquitto Broker läuft auf einer virtuellen Maschine unter Linux.
Die Publisher veröffentlichen in gegeben Zeitintervallen die Daten unter einem bestimmten Topic.
Die Subscriber beobachten dieses Topic um neue Daten zugesendet zu bekommen.
Zum Erheben der Messdaten wir ein eigens in C++ geschriebenes Programm verwendet, welches jede Sekunde die Auslastung des Arbeitsspeichers sowie der CPU ausließt.

Auswirkung der Nachrichtengröße

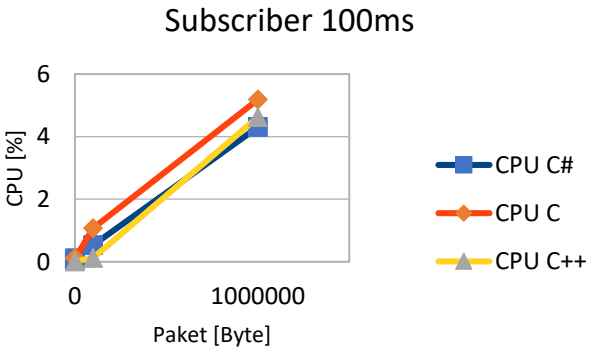
Bei dieser Messreihe werden Nachrichten verschiedener Größe bei einem Intervall von 100 ms veröffentlicht. Die Nachrichtengröße beträgt 0, 1.000, 100.000 und 1.000.000 Bytes.

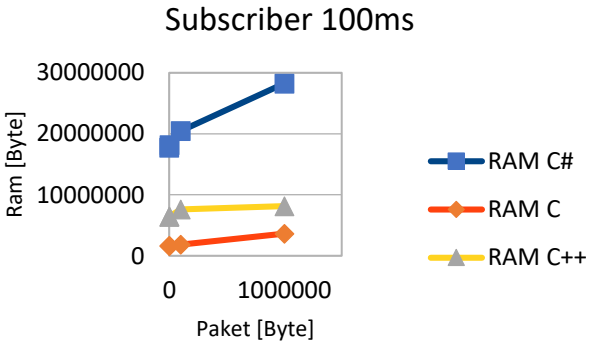
Für die Auswertung wurde zwei Minuten lang die CPU und RAM Auslastung gemessen und jeweils der Mittelwert gebildet.





Auf den Abbildungen der Publisher ist zu erkennen, dass bei C# die größte RAM Auslastung ist, die mit steigender Nachrichtengröße auch weiterwächst. 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 das in beiden Fällen die Paho MQTT Library genutzt wird.
Die hohe RAM Auslastung bei C# lässt sich durch den durch C# erzeugten Overhead erklären.

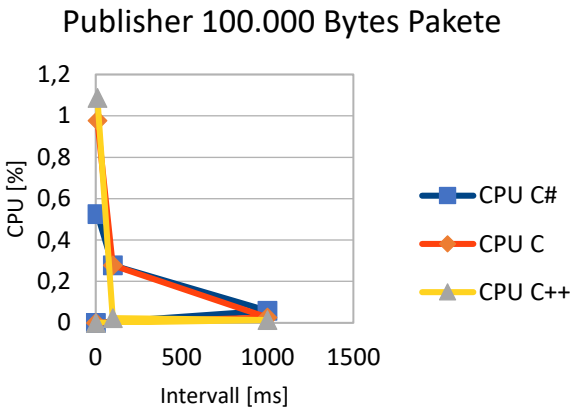


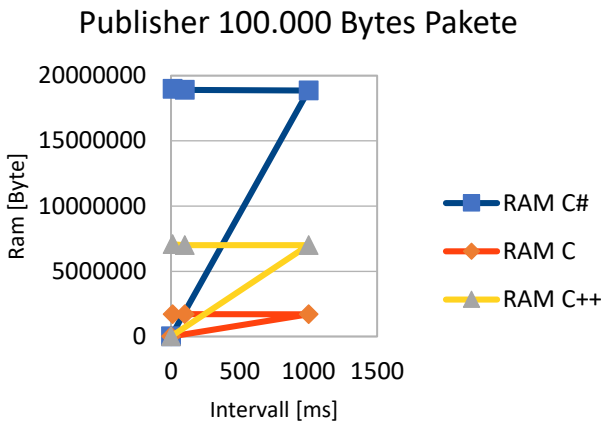


Auch bei den Subscribern ist ein erhöhter
RAM Bedarf beider C# Implementation zu sehen.
Die CPU Auslastung steigt hier jedoch bei allen drei Implementationen stark an mit steigender Paketgröße. Der Unterschied zu den Publishern ist in diesem Fall, das die Publisher bei Programmstart initial den Speicher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

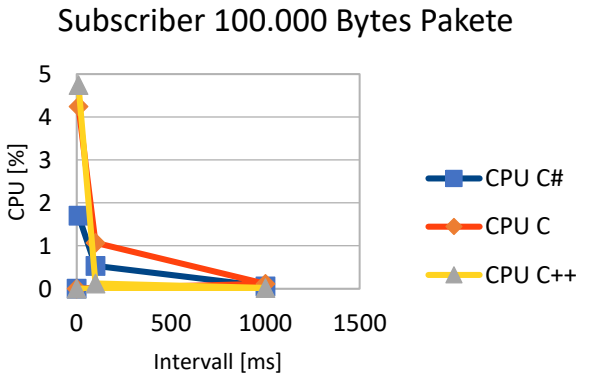
Auswirkungen des Publishingintervalls

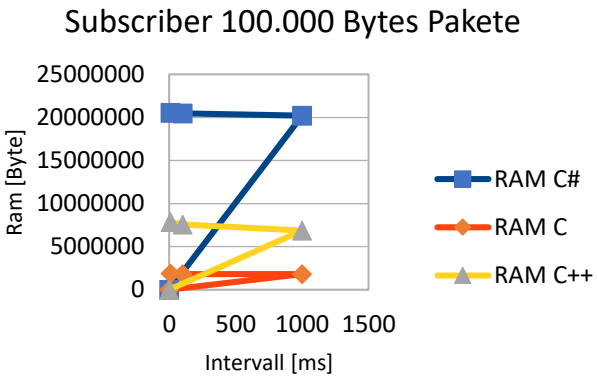
Als nächstes wird die Auswirkung des Publishingintervalls bei fester Datenmenge gemessen. Dazu wird eine Nachrichtengröße von 100.000 Bytes in den Intervallen 10, 100 und 1000 ms gesendet. Es wird jeweils der Durchschnitt aus zwei Minuten gewertet.





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

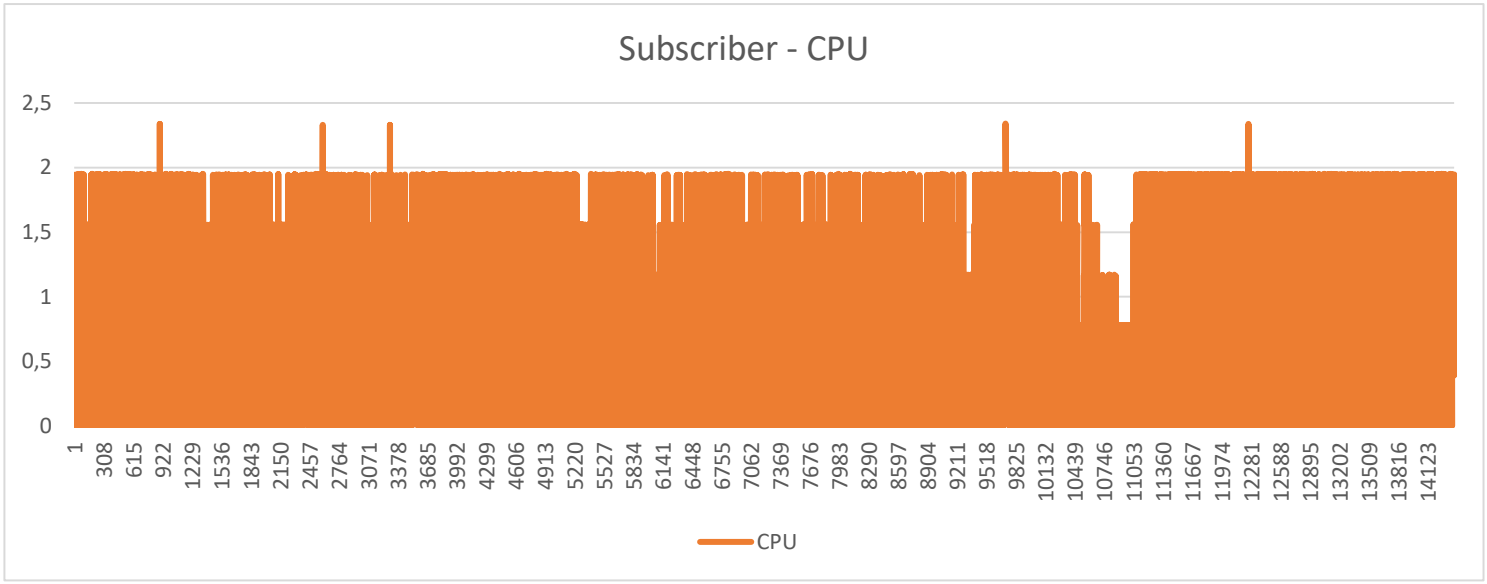


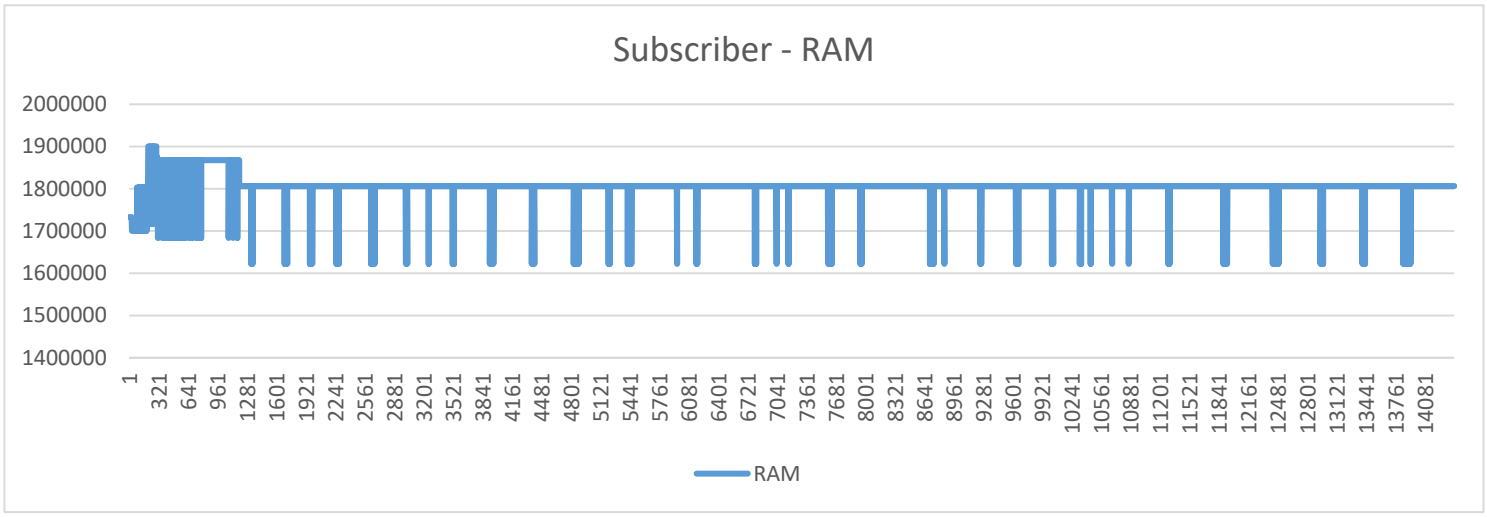


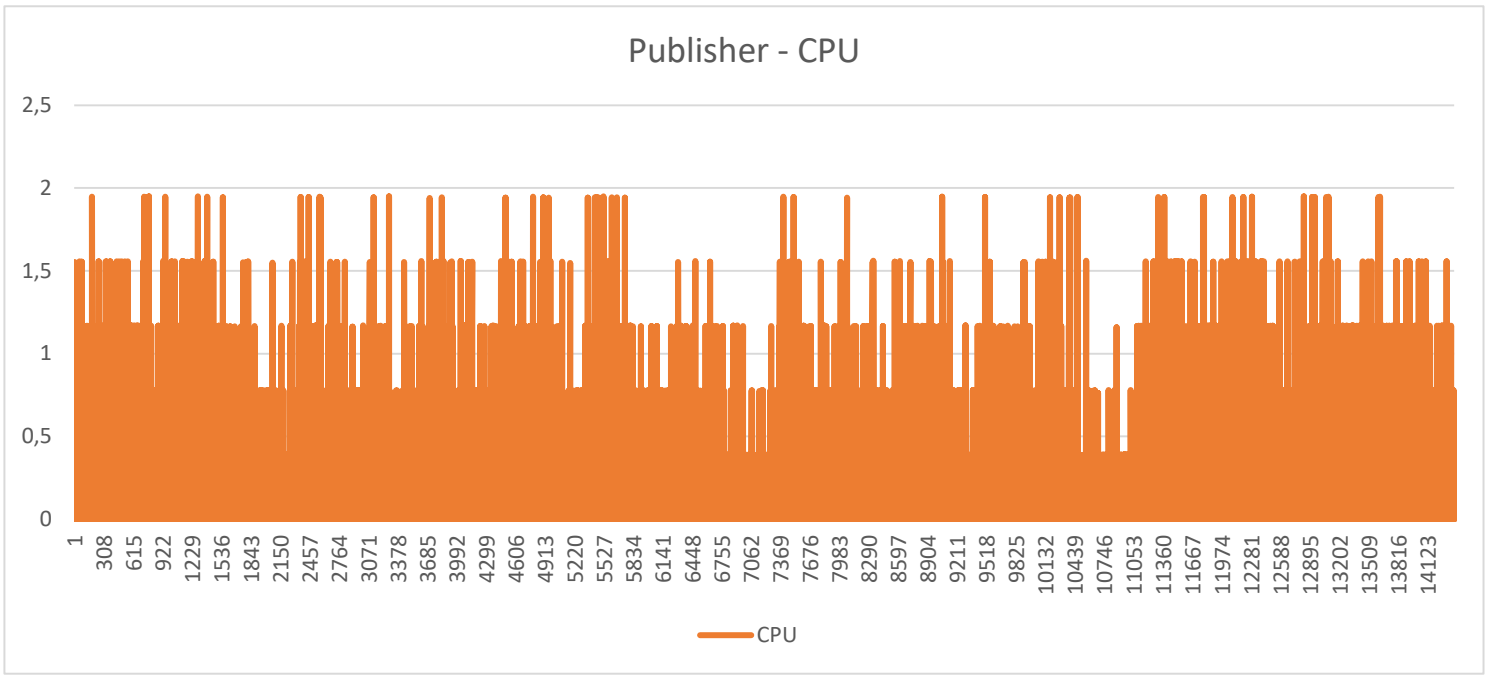
Auch bei den Subscribern ist eine sinkende CPU Auslastung bei höheren Frequenzen zu sehen. Die RAM Auslastung ist bei allen Implementationen bei allen Intervallen nahezu gleichbleibend.

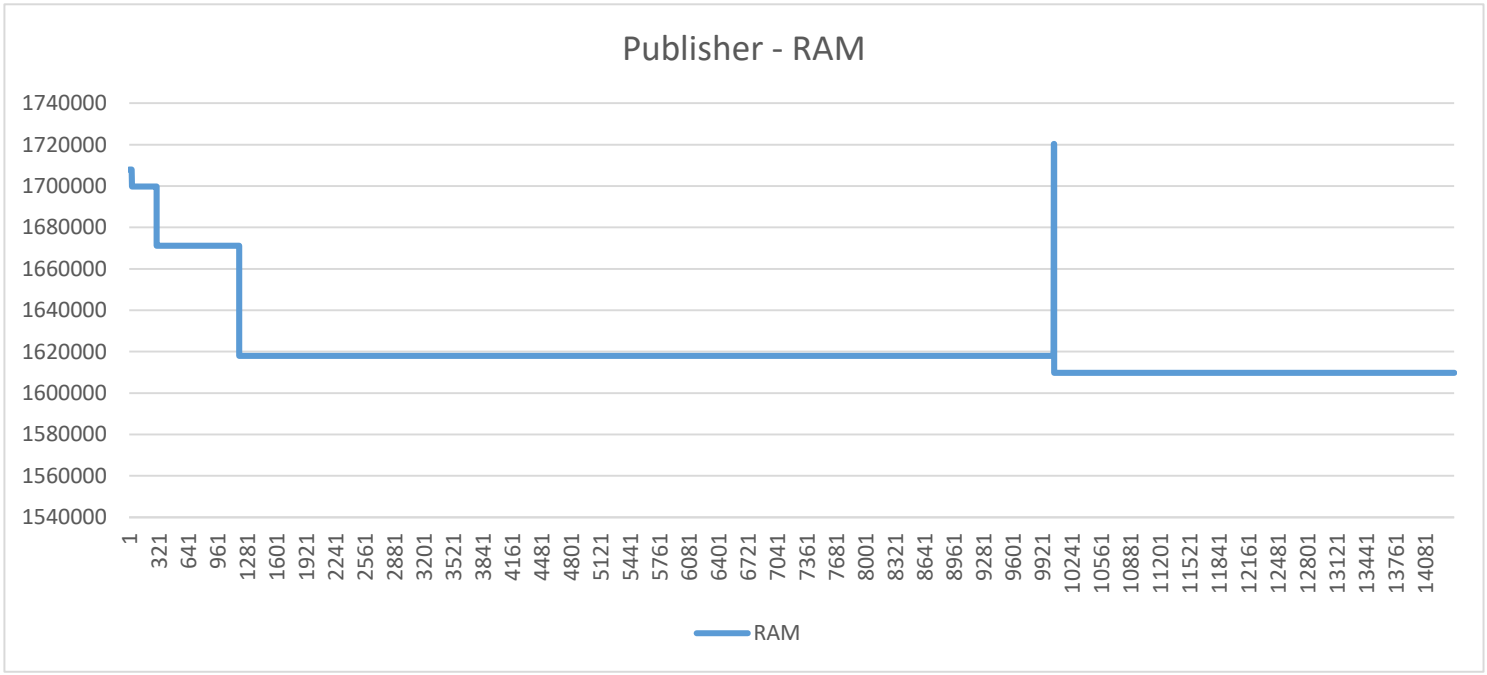
Langzeitmessungen

C:



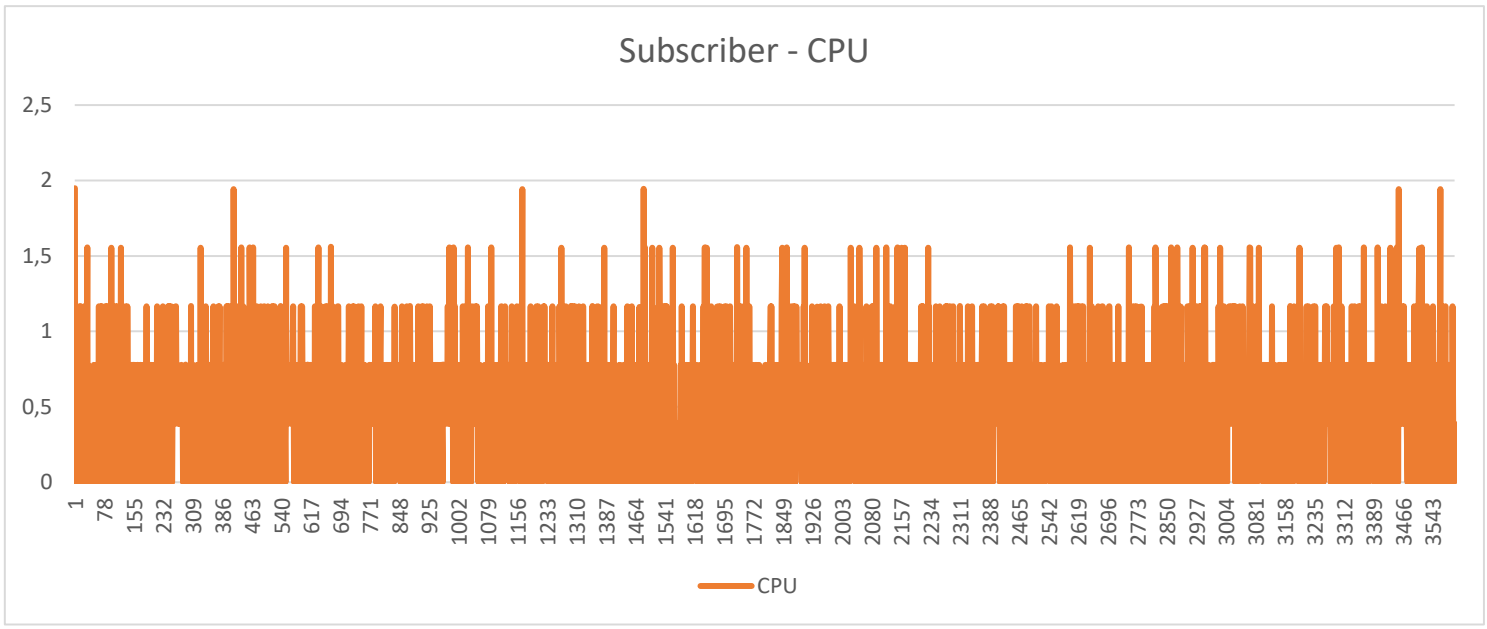


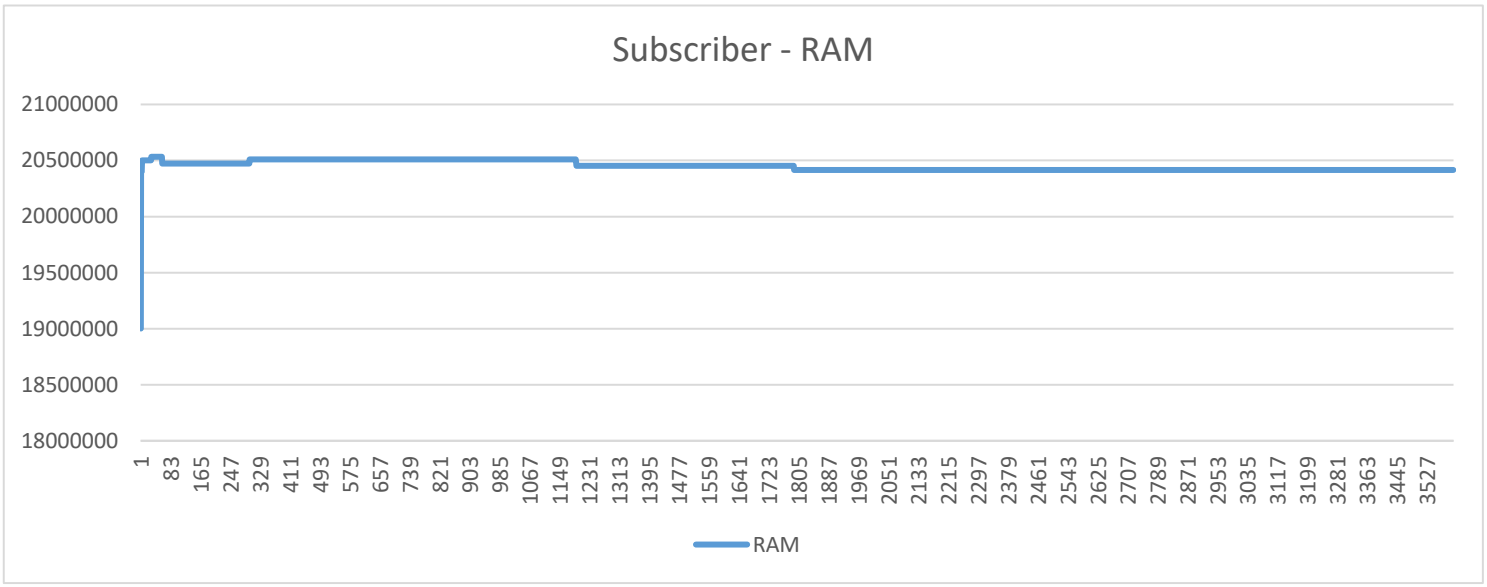


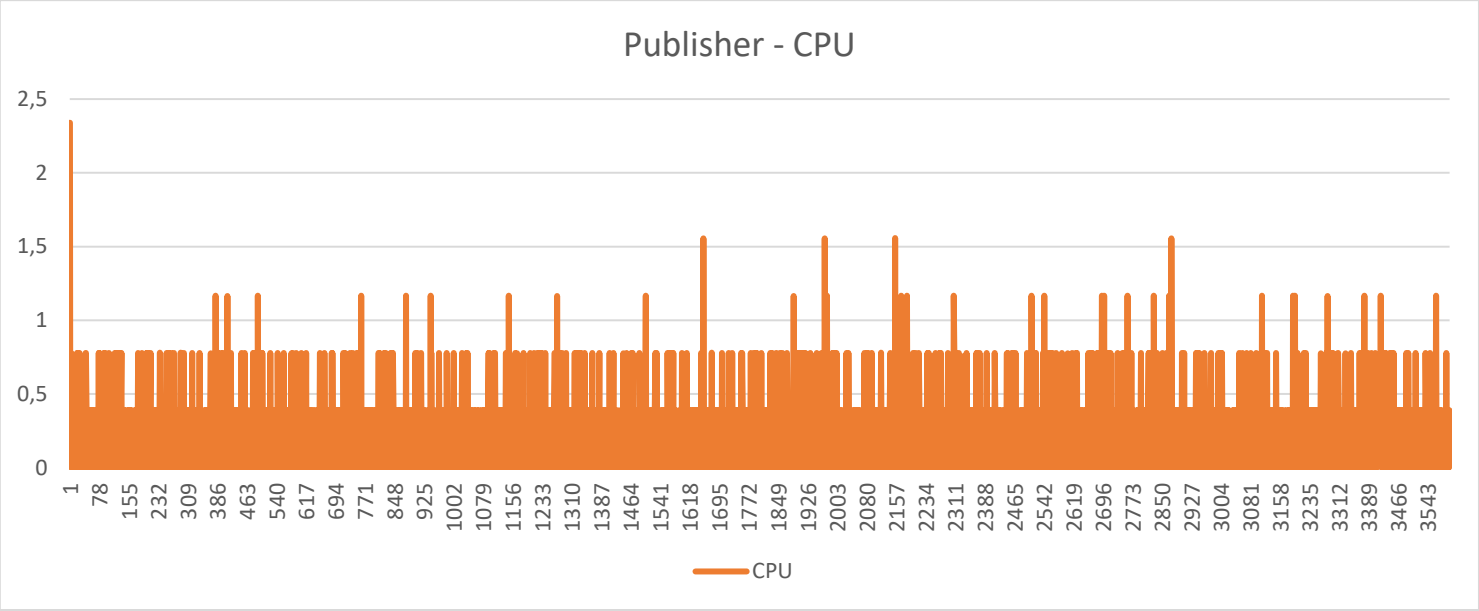


Bei den Langzeitmessungen in C ist zu erkennen, dass sich der RAM und CPU Bedarf relativ stabil verhält. Es ist erkennbar, dass sich die RAM-Auslastung beim Subscriber in bestimmten Zeitintervallen kurzzeitig verringert.

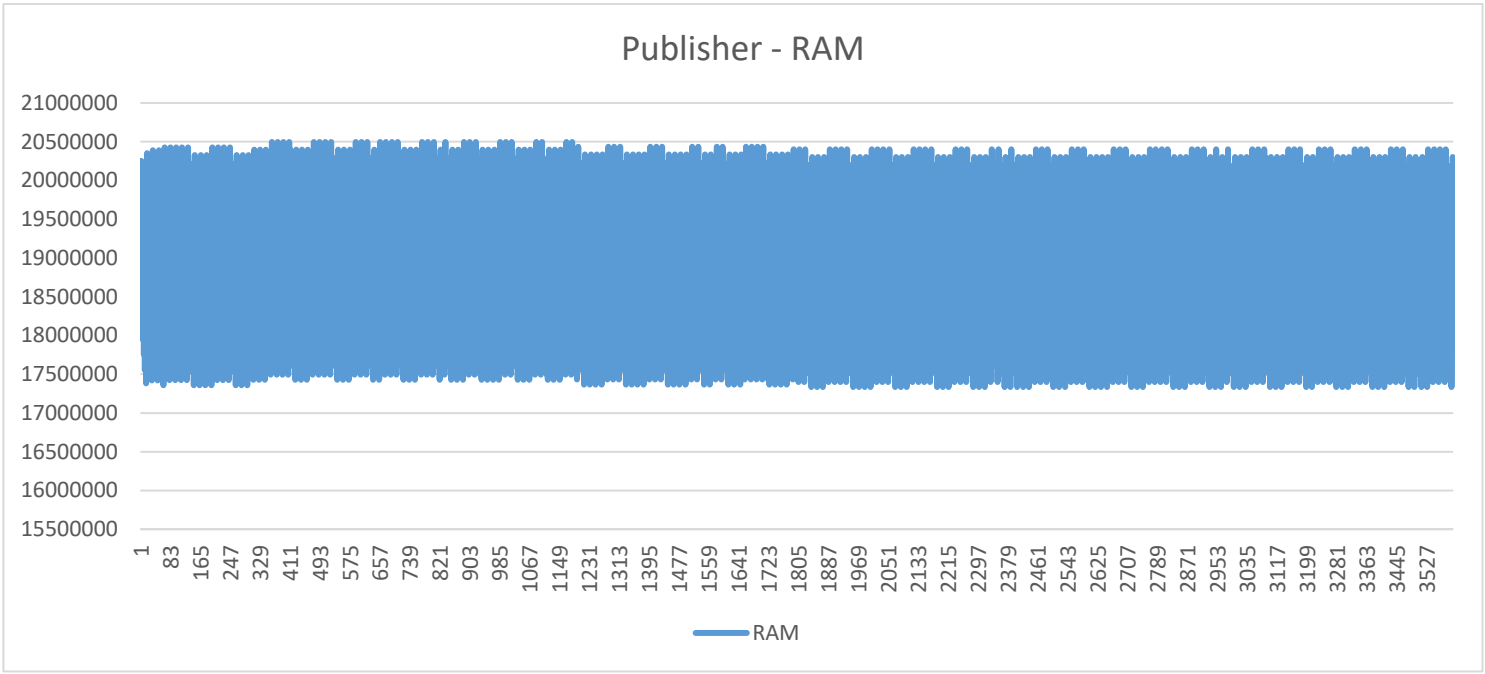
C#:


Subscriber - CPU
2,5
2
1,5
1
0,5
0
1
78
155
232
309
386
463
540
617
694
771
848
925
1002
1079
1156
1233
1310
1387
1464
1541
1618
1695
1772
1849
1926
2003
2080
2157
2234
2311
2388
2465
2542
2619
2696
2773
2850
2927
3004
3081
3158
3235
3312
3389
3466
3543
CPU

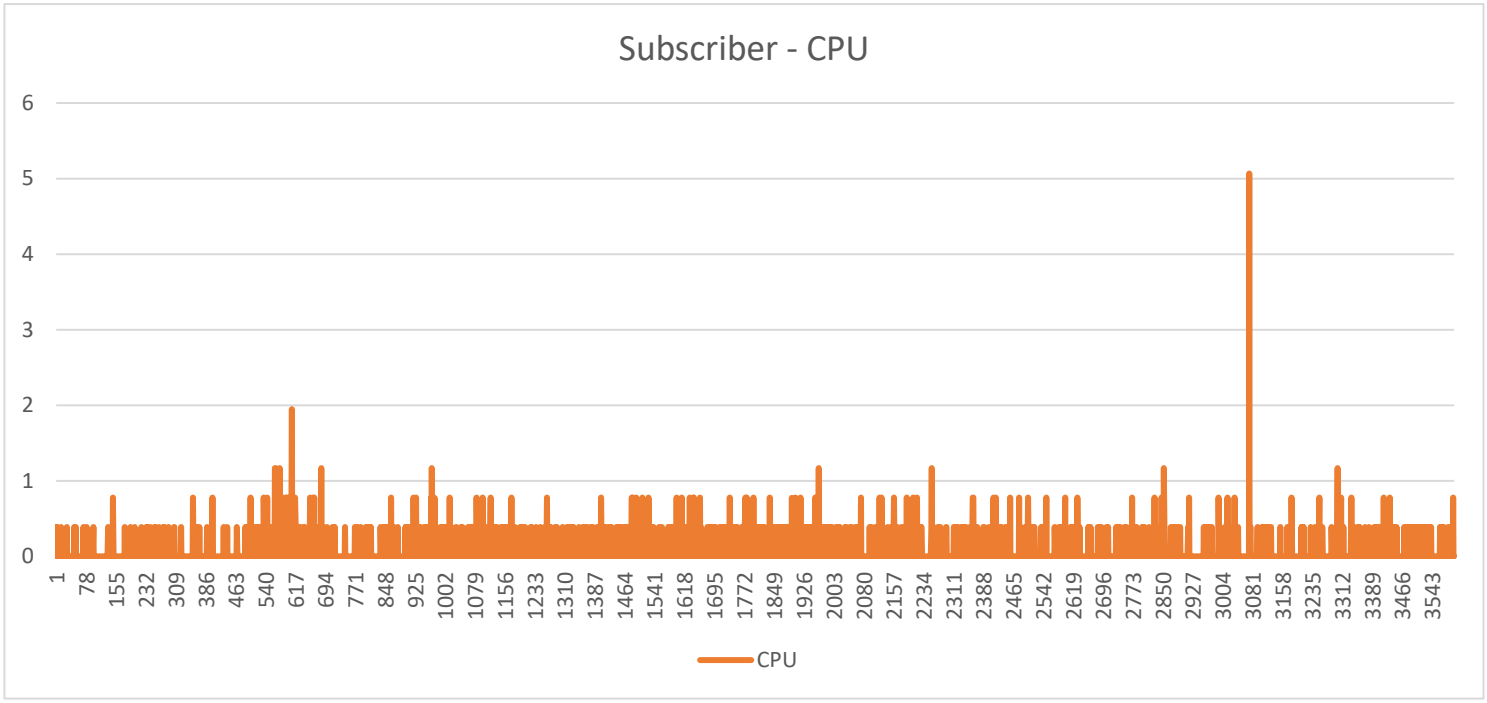

Subscriber - RAM
21000000
20500000
20000000
19500000
19000000
18500000
18000000
1
83
165
247
329
411
493
575
657
739
821
903
985
1067
1149
1231
1313
1395
1477
1559
1641
1723
1805
1887
1969
2051
2133
2215
2297
2379
2461
2543
2625
2707
2789
2871
2953
3035
3117
3199
3281
3363
3445
3527
RAM

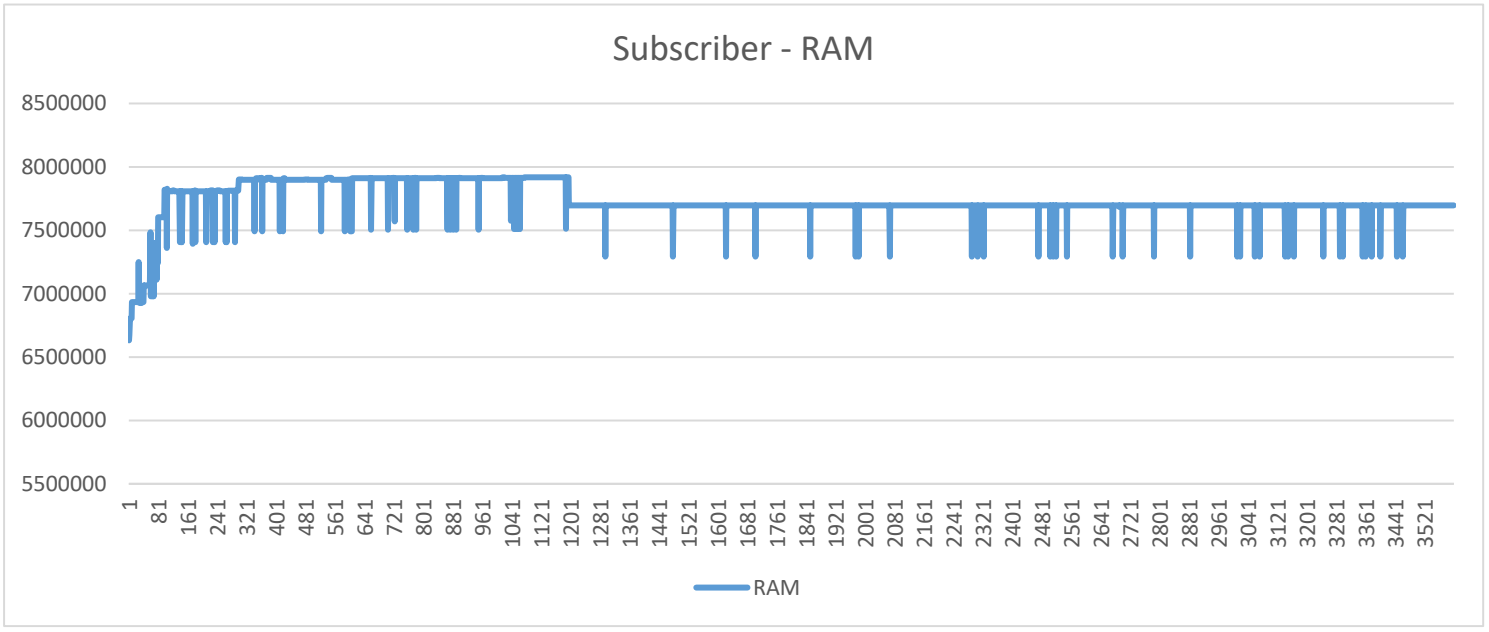


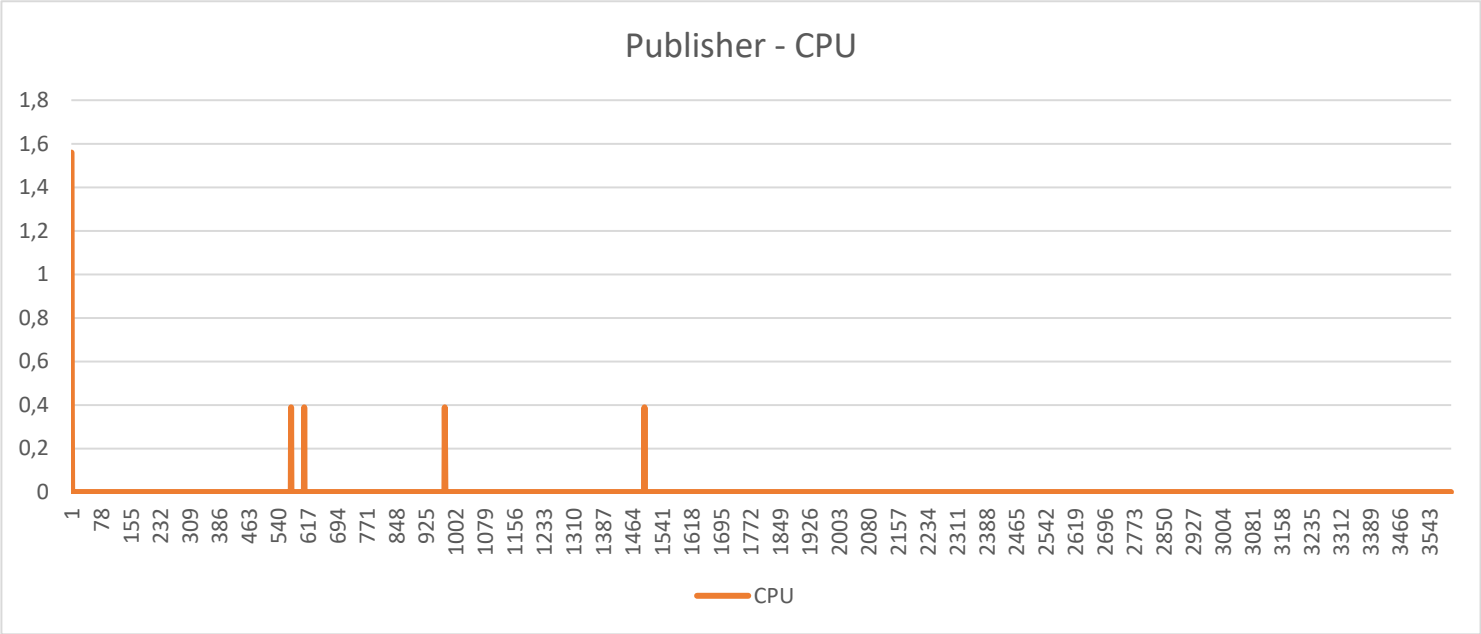
Auch bei der C# Implementation ist kein dauerhafter Anstieg an CPU oder Arbeitsspeicherbedarf zu vermerken. Es fällt jedoch auf, dass der Garbage Collector beim Publisher deutlich sichtbar arbeitet.

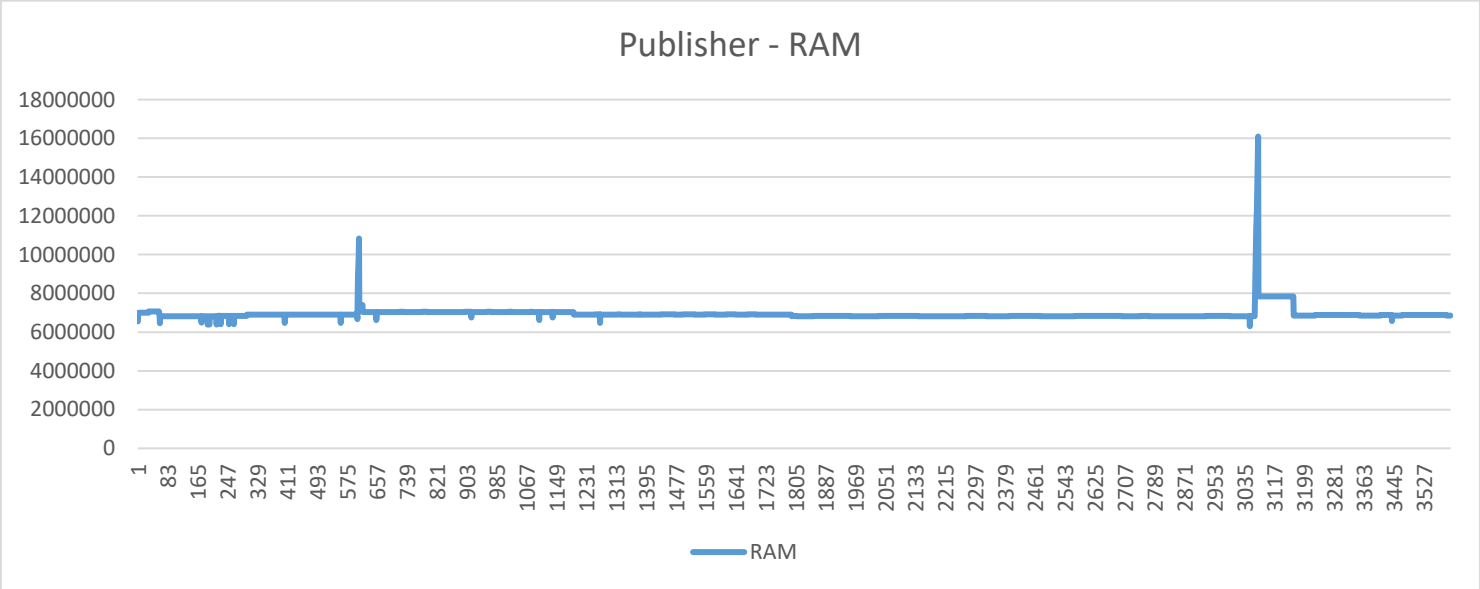


C++:


Subscriber - CPU
6
5
4
3
2
1
0
1
78
155
232
309
386
463
540
617
694
771
848
925
1002
1079
1156
1233
1310
1387
1464
1541
1618
1695
1772
1849
1926
2003
2080
2157
2234
2311
2388
2465
2542
2619
2696
2773
2850
2927
3004
3081
3158
3235
3312
3389
3466
3543
CPU


Subscriber - RAM
8500000
8000000
7500000
7000000
6500000
6000000
5500000
1
81
161
241
321
401
481
561
641
721
801
881
961
1041
1121
1201
1281
1361
1441
1521
1601
1681
1761
1841
1921
2001
2081
2161
2241
2321
2401
2481
2561
2641
2721
2801
2881
2961
3041
3121
3201
3281
3361
3441
3521
RAM





Auch die C++ Langzeitmessungen laufen am stabilsten. Es fällt auf, dass der Subscriber nahezu keine CPU-Auslast benötigt.

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)

Anleitung:

Sämtliche Anwendungen werden über die Konsole gestartet.

C:

```
Publish: filename.exe <pub> <clientid> <payloadLen> <delay>
Subscribe: filename.exe <sub> <clientid>
```

C#:

```
Publish: filename.exe <payloadLen> <delay>
Subscribe: filename.exe
```

C++:

```
Publish: filename.exe <payloadLen> <delay>
Subscribe: filename.exe
```