HIVEMQ

ENTERPRISE MQTT BROKER

TLS BENCHMARKS

HiveMQ 3.1.0 on AWS

03/2016

Introduction

HiveMQ is a highly scalable Enterprise MQTT Broker designed for lowest latency and very high throughput while supporting state-of-the-art security. This benchmark document compares key runtime metrics of HiveMQ for MQTT + TLS 1.2 with runtime metrics of HiveMQ with MQTT over TCP.

The goal of this document is to explore how the use of TLS 1.2 affects the performance and resource consumption of HiveMQ. The servers used in the benchmark scenarios are typical servers HiveMQ customers use on a day-to-day basis and there are no obscure settings applied which could falsify the results. HiveMQ is installed with the default configuration and while there are many performance relevant knobs available in HiveMQ, the benchmarks were executed without any optimization. All Quality of Service benchmarks use disk persistence so all guarantees the MQTT specification requires are in place.

INFO: This is a technical document and it's assumed that the reader is familiar with the basic principles and concepts of MQTT and TCP/IP.

Benchmark Environment

All benchmarks were executed on *Amazon Web Services (AWS)*, a cloud infrastructure provider.

AWS allows to deploy servers on a shared environment and is often used for cloud services. AWS is by far the most popular cloud provider of HiveMQ customers, so using AWS for the benchmark was a natural choice.

Important: Virtual Machines by definition can't be as performant as physical hardware. If multiple VMs share the same hardware, the CPU utilization is not necessarily as expressive as on real hardware. No dedicated hardware servers were used in this benchmark and no AWS dedicated instances were used.

Benchmark Scenarios

Each of the benchmarks shows the resource consumption in terms of CPU and Java heap space when using MQTT with and without **TLS 1.2.** The benchmark was not designed for stress testing MQTT clients or the broker. It was created to learn about the behaviour and performance differences when TLS is in place.

The benchmark uses one server instance of the HiveMQ Benchmark Tool running on a EC2 instance and HiveMQ, running on a separate EC2 instance. Both servers were hosted in the same AWS availability zone.

OpenJDK 1.7.0_79 was used in these benchmarks.

The RSA X509 server certificate had a key size of **4096-bit**.

Hardware

HiveMQ Server Instance

The following EC2 Instance Type was used for the HiveMQ installation:

HiveMQ EC2 Instance Details

Name	Value
Instance Type	C4.2xlarge
RAM	15GiB (~16GB)
vCPU	8
Physical Processor	Intel Xeon E5-2666 v3
Clock Speed (GHz)	2.9

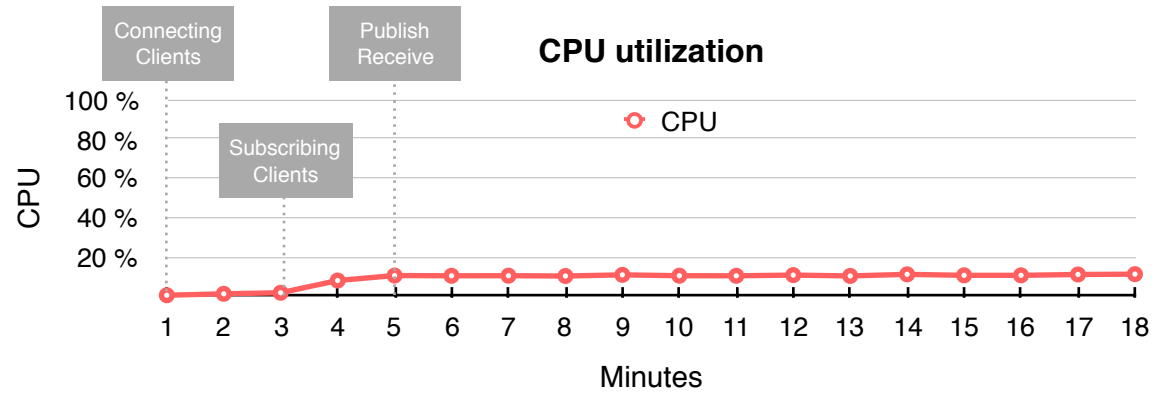
The *c4.2xlarge* Instance Type is an AWS EC2 Instance for computing intensive applications and has 8 (virtual) cores.

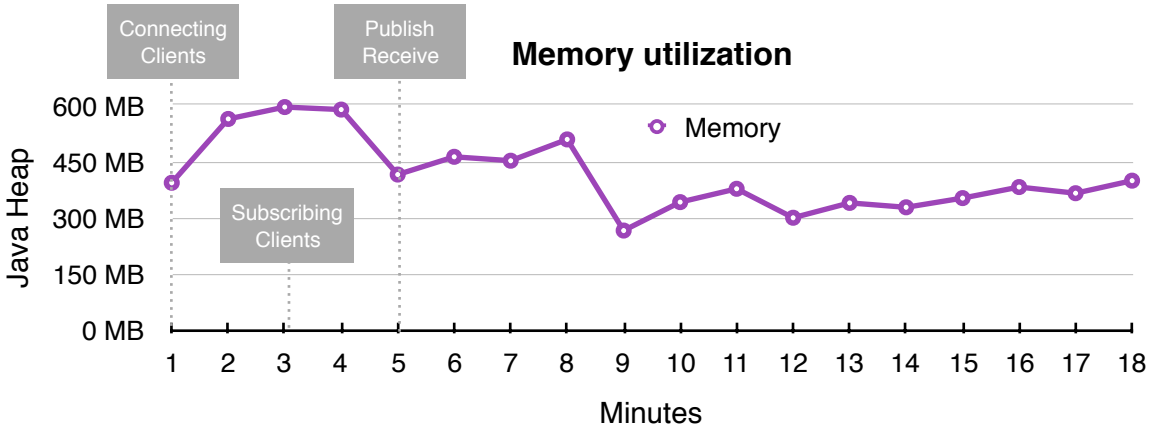
Important: All EC2 instances in this benchmark are shared instances, not dedicated instances, which means the CPU steal time is significantly higher than on dedicated instances. This is the most common deployment type we saw from customers, so the goal was not to artificially improve the results by using an uncommon deployment setup.

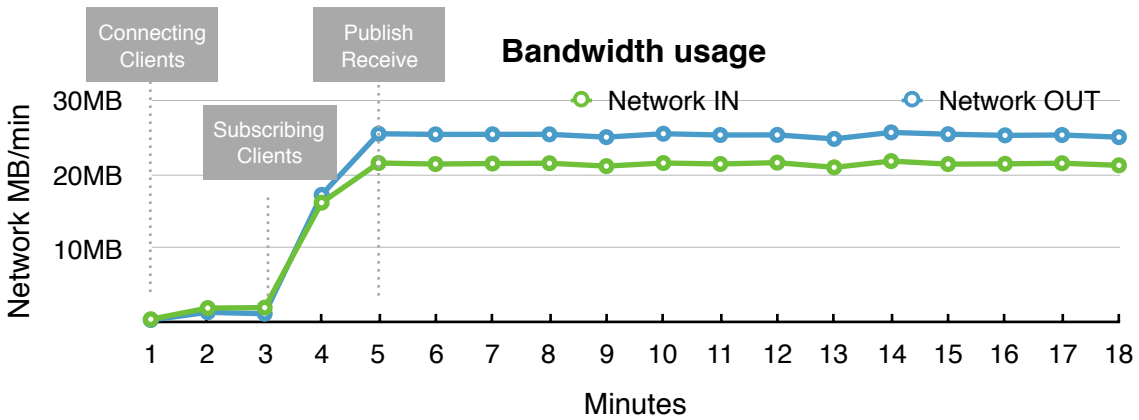
10.000 MQTT Clients without TLS

Test Parameters	
MQTT Clients	10.000
Messages / Second	1.000
Connections / Second	100
Quality of Service Level	1

This benchmark spawns 10.000 real MQTT clients, connects the MQTT clients in batches of 100 clients per second without using TLS, subscribes all MQTT clients to a unique topic per client and finally each client publishes one message in 10 seconds.





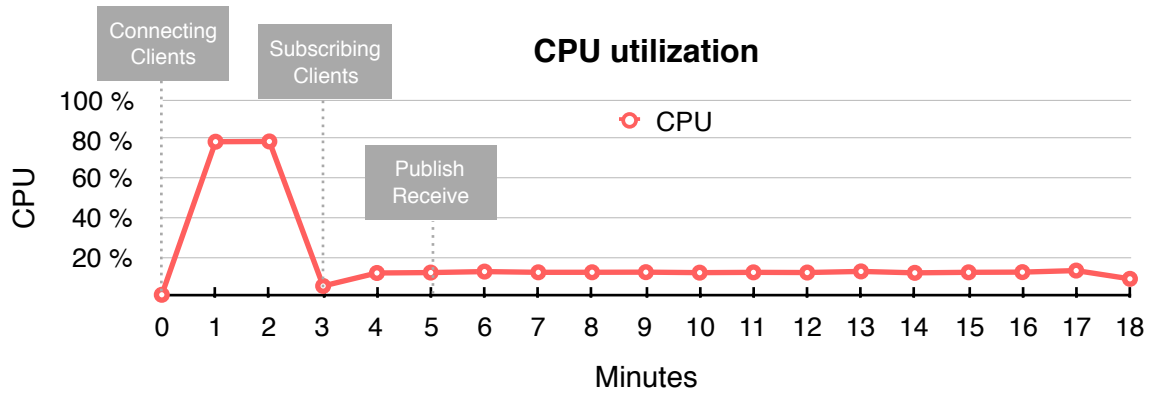


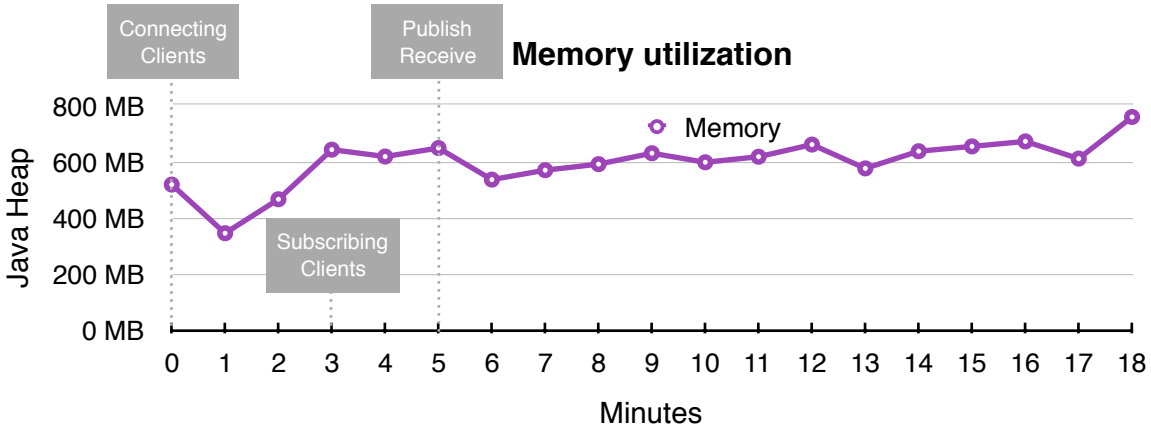
10.000 MQTT Clients with TLS

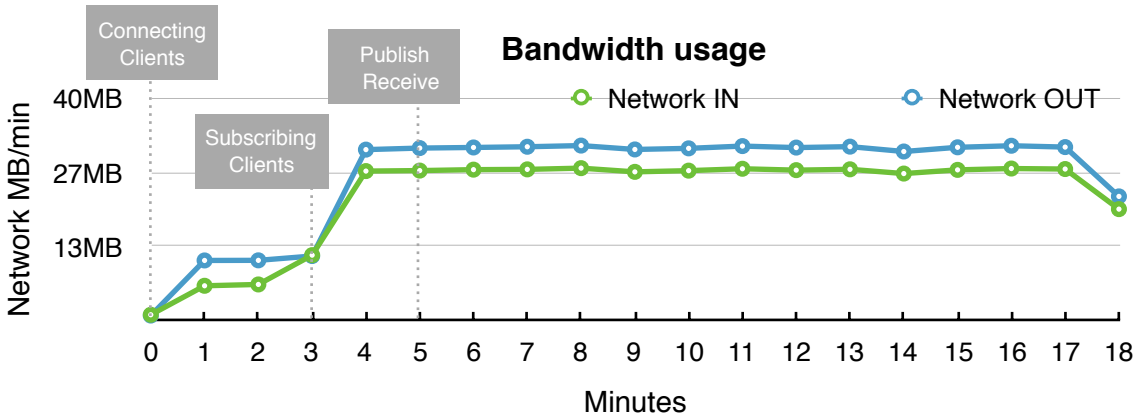
Test Parameters	
MQTT Clients	10.000
Messages / Second	1.000
Connections / Second	100
Quality of Service Level	1

This benchmark spawns 10.000 real MQTT clients, connects the MQTT clients in batches of 100 clients per second using TLS, subscribes all MQTT clients to a unique topic per client and finally each client publishes one message in 10 seconds.

The cipher suite used for this benchmark was **TLS_RSA_WITH_AES_128_CBC_SHA**



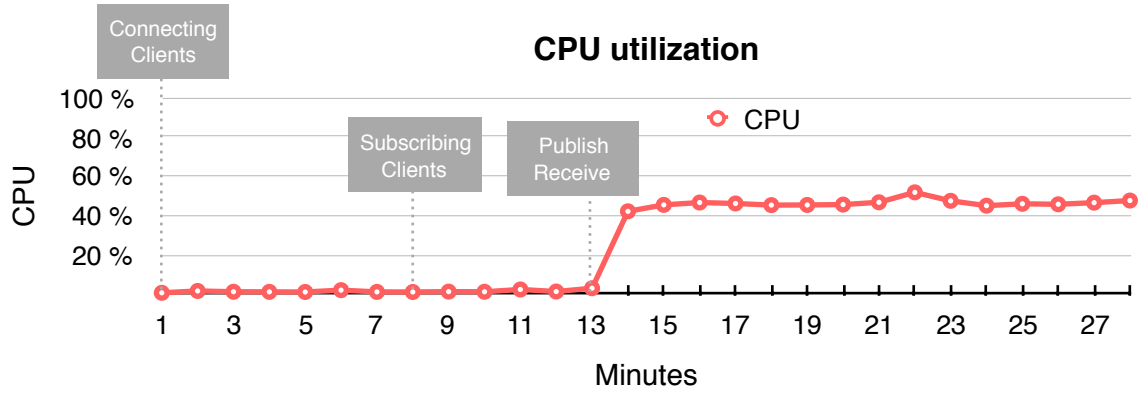


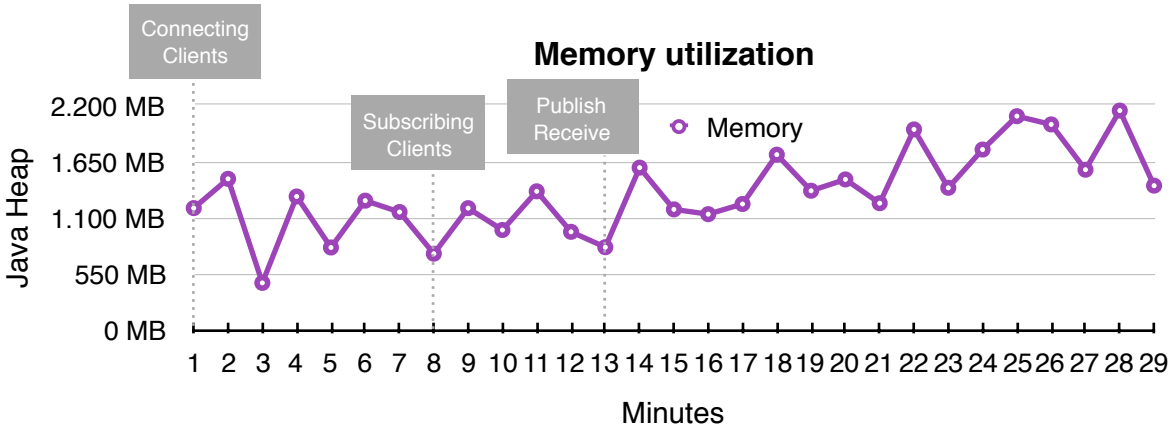


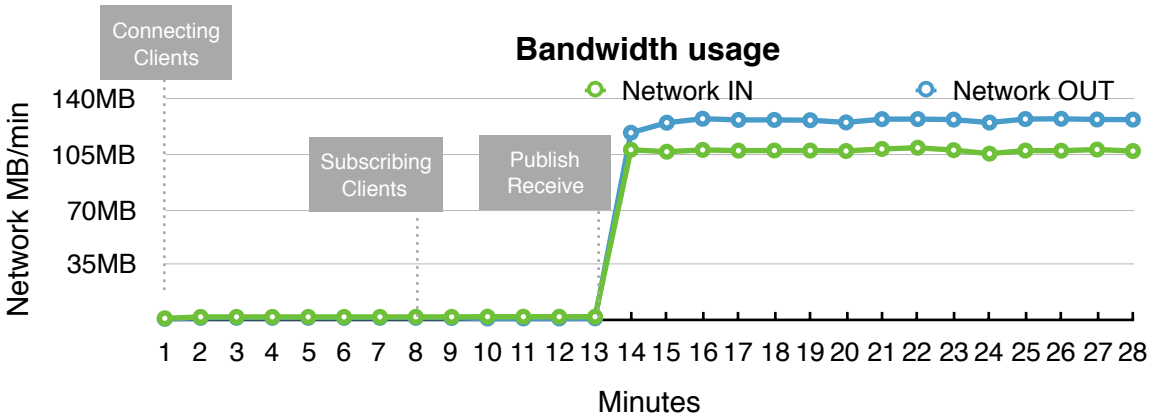
50.000 MQTT Clients without TLS

Test Parameters	
MQTT Clients	50.000
Messages / Second	5.000
Connections / Second	100
Quality of Service Level	1

This benchmark spawns 50.000 real MQTT clients, connects the MQTT clients in batches of 100 clients per second without using TLS, subscribes all MQTT clients to a unique topic per client and finally each client publishes one message in 10 seconds.





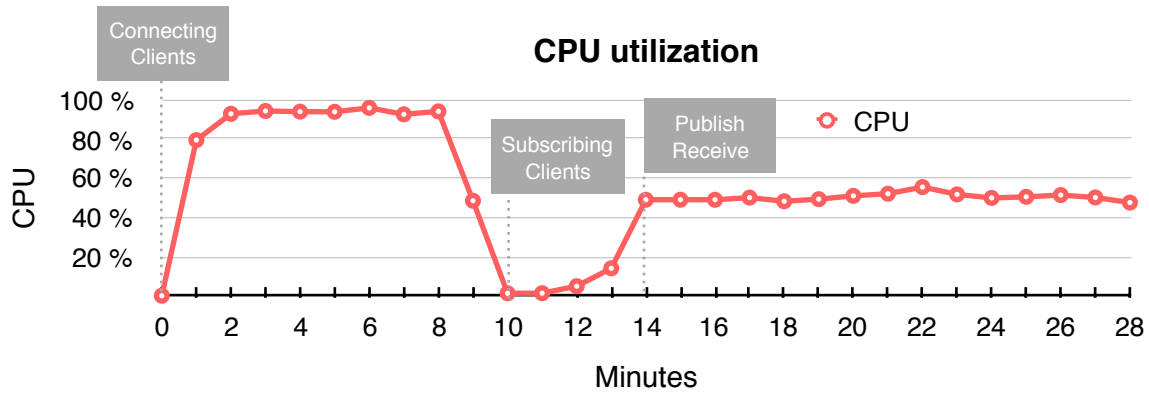


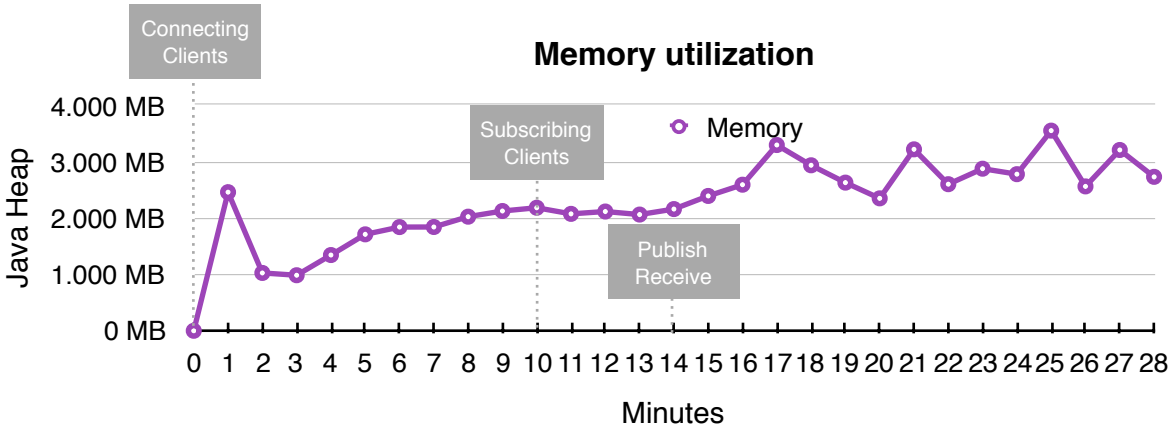
50.000 MQTT Clients with TLS

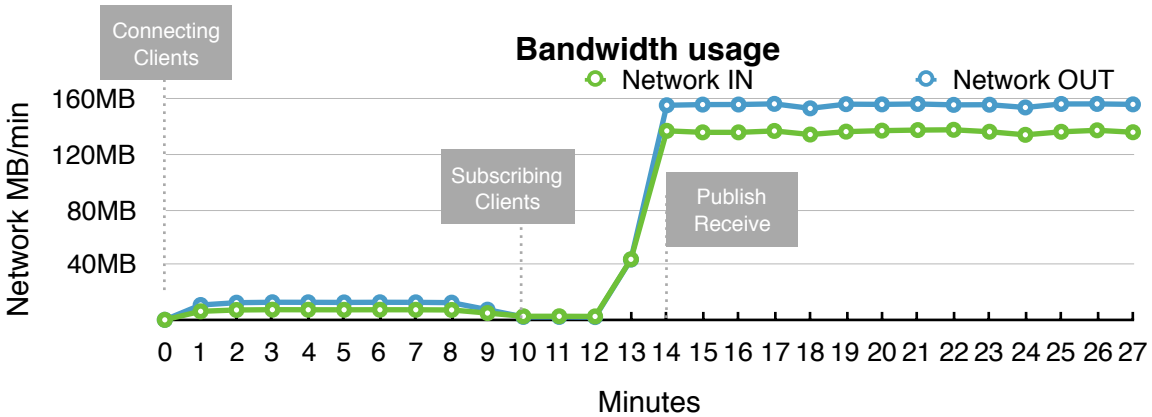
Test Parameters	
MQTT Clients	50.000
Messages / Second	5.000
Connections / Second	100
Quality of Service Level	1

This benchmark spawns 50.000 real MQTT clients, connects the MQTT clients in batches of 100 clients per second using TLS, subscribes all MQTT clients to a unique topic per client and finally each client publishes one message in 10 seconds.

The cipher suite used for this benchmark was **TLS_RSA_WITH_AES_128_CBC_SHA**







Summary & Discussion

The benchmarks demonstrated that the use of TLS comes with a cost in terms of resource consumption. The TLS handshake when connecting 100 MQTT clients per second is rather expensive in terms of CPU. The use of 4096-bit certificates caused - in addition to the TLS_RSA_WITH_AES_128_CBC_SHA cipher suite - additional overhead compared to lower key sizes. Using a 2048-bit instead of a 4096-bit certificate could reduce CPU load for the TLS handshake by factor 2x-5x.

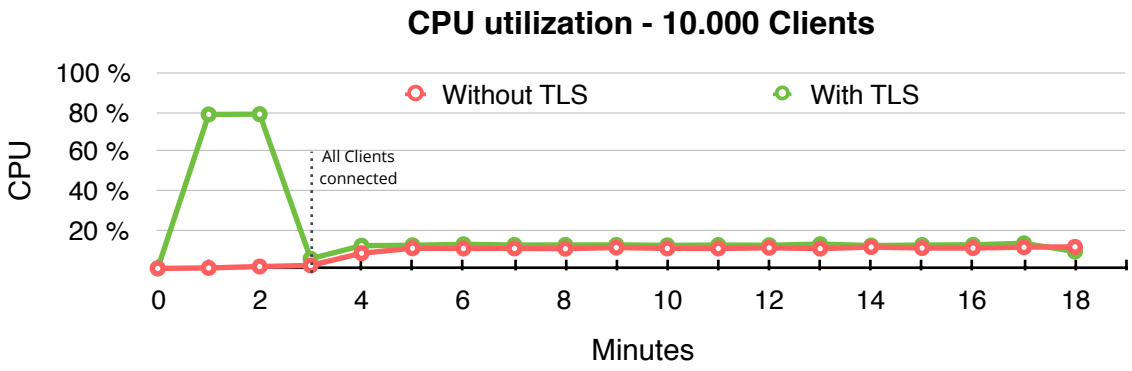
The average HiveMQ memory usage increased when using TLS by factor 1.1x - 1.8x. The additional memory is caused by the need for allocating additional buffers for TLS, so this needs to be taken into account for sizing production servers correctly.

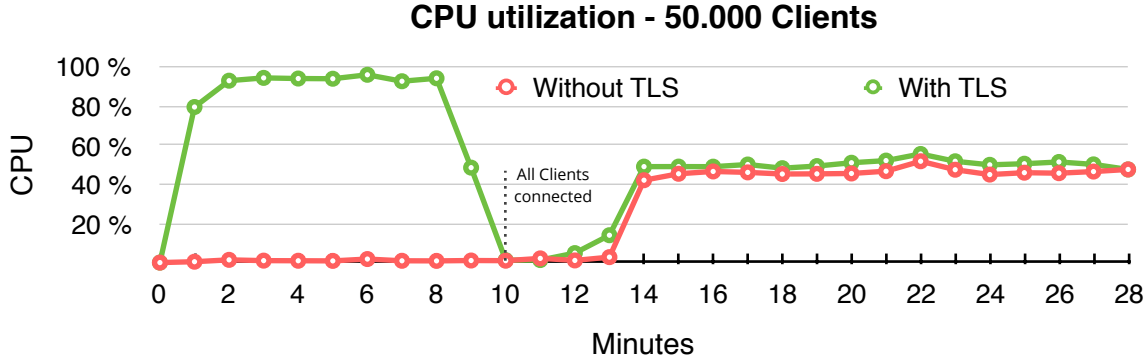
The total incoming and outgoing bandwidth usage increased when using TLS, which is an expected result since a CBC cipher suite (TLS_RSA_WITH_AES_128_CBC_SHA) was used.

CPU Comparisons

The following charts compare the CPU utilization for scenarios with and without TLS.

The CPU overhead for the TLS handshake is significant and the CPU usage shows a clear spike — only while initially connecting MQTT clients — compared to the scenarios without TLS. While noticeable, the CPU consumption for TLS when clients are publishing and subscribing is not significant.





MQTT and TLS

MQTT uses long-living TCP connections, which means the TLS handshake takes place — in the best case — once in the lifetime of a MQTT client. The benchmark demonstrated that the TLS overhead is very small and almost negligible once the TLS handshake finished. If the MQTT broker faces frequent client reconnects and thus the TLS handshake consumes too much CPU, the following alternatives could be considered:

- Using a Load Balancer with SSL Termination. Many Load Balancers support SSL offloading which can increase performance significantly.
- SSL Session Resumption with Session IDs can be used to avoid a complete TLS handshake when a MQTT client reconnects. Bear in mind that session resumption is most secure when using cipher suites that have Perfect Forward Secrecy characteristics.
- X509 certificates with smaller key length (e.g. 1024-bit or 2048-bit) could be used.
- Elliptic Curve Cryptography (ECC) could potentially reduce CPU consumption

> Different TLS Cipher Suites may have different performance characteristics. Our recommendation is to stick with the most secure cipher suites available for your MQTT clients and your JVM, especially if you are planning to use SSL Session Resumption.

Conclusion

Using TLS 1.2 can dramatically improve security of the MQTT system. The tradeoff for using TLS 1.2 with MQTT was explored in this benchmark. The tests revealed that the TLS handshake is by far the most expensive operation, especially with high X509 certificate key lengths. The TLS overhead once a client is connected is negligible. The fact that MQTT uses long-living TCP connections make TLS and MQTT a natural choice for secure messaging deployments as the TLS handshake is essentially a cost to pay once (instead of with every request when using e.g HTTPS) and the runtime overhead of TLS is insignificant. The use of SSL Session resumption, which is supported by HiveMQ without further configuration, reduces the TLS Handshake overhead even further.

The benchmark demonstrated that the Enterprise MQTT Broker *HiveMQ* still delivers stellar performance when using state-of-the-art security mechanisms like TLS 1.2 with 4096-bit X509 certificates.

HiveMQ is the MQTT broker for the connected enterprise:
The puzzle piece between constrained devices and enterprise systems.
HiveMQ is scalable, secure and simple

http://www.hivemq.com