# SEMA Cloud Packet Loss Analysis

## Goal

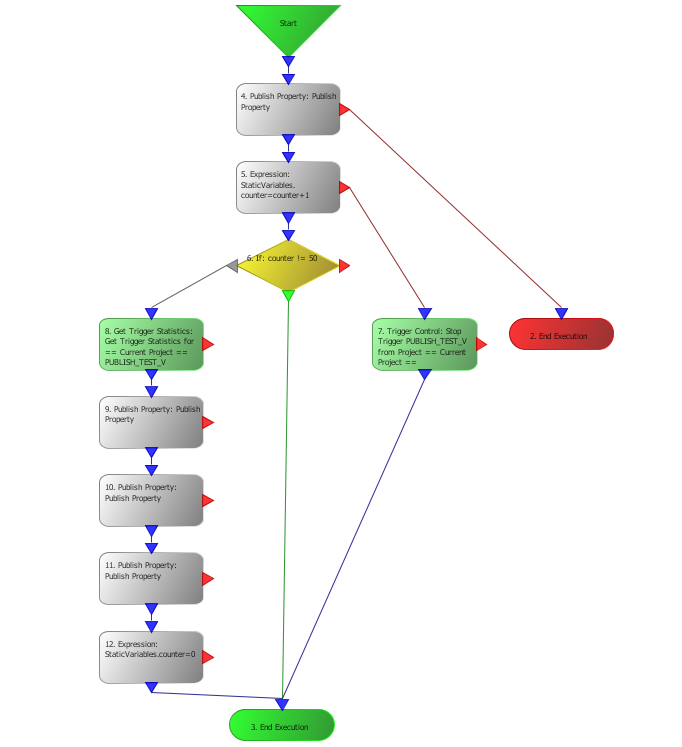
The target of this analysis was to demonstrate if a large number of clients is able to connect to the cloud servers and transmit data and to quantize eventual packet loss. As a large number of physical clients is hard to manage, a virtualized approach was chosen.

## Setup

The packet drop analysis tests were run on an ADLINK NuPRO-A40h board, equipped with an Intel Core i7-3770CPU (four Cores + Hyper threading, totaling eight virtual cores at 3.40 GHz each) and 16GB DDR3-1600-RAM. The operating system is a Debian7 based Linux distribution with Kernel 3.7.9, 32 bit with PAE (physical address extension). The Gateway software is DeviceWise Gateway 14.2.0-011 for Linux X86 (DWGateway\_Install.Linux-X86-Generic.14\_2\_0-011.tar.gz). The test server is “api-de.devicewise.com” located at Frankfurt, Germany.

## Test procedure

To simulate a high number of instances reporting to the cloud simultaneously, the gateway was modified to run one “master” instance and 488 “swarm” instances. This approach emulates 489 independent gateway installations with unique thing keys. Each thing key is derived from the board’s MAC primary address by appending the instance number to the MAC address string (e.g. the thing key for the master instance is 00306422ce3b, the thing key for the 254th instance is 00306422ce3b254). Each instance runs a single trigger which transmits an INT4 property value. To make individual trigger events identifiable and to make data loss detectable, the transmitted value is incremented after transmission. When this counter reaches 50, the value is reset to 0 and additional trigger statistics (counters for success, failure and overflow) are published as separate properties and the process starts all over again.



Since each instance incorporates a separate database and these are very sensitive to disk read/write latencies, the affected directories where the local databases reside were moved to a ram disk to eliminate this possible performance hazard. The required ram disk space for the gateway instances and the associated databases was 531MB, memory utilization for all processes is about 2GB and the average CPU load was approx.. 10-15%. Negative influences from high system load were not expected under this conditions.

After the swarm was run for 30 hours, the failure and overflow counters were collected from all instances.

During the test run, a total number of 55346976 property values were published. This number is composed by 52812000 “normal” property publish events ( 489 instances \* 30h \* 3600s/h) and 2534976 “statistics” property publish events (489 instances \* 30h \* 3600s/h \* 3 properties every 50 seconds).

## Test results

### “Failure” counter

On 67 out of 489 instances , the failure counter was increased during the test run. This means that 13.7% of the instances reported at least one failure. The highest individual fail count was 4 (four). For the distribution of number of failures over the number of instances see table and graph below:

|  |  |  |
| --- | --- | --- |
| **Number of failures** | **Affected number of instances** | **Percentage** |
| 0 | 422 | 86.3% |
| 1 | 36 | 7.4% |
| 2 | 27 | 5.5% |
| 3 | 3 | 0.6% |
| 4 | 1 | 0.2% |
| >4 | 0 | 0% |

Set into relation to the total number of published properties (55346976 publish events), every 537349th event resulted in a failure, equaling a failure quote of 1.86e-6.

### “Overflow” counter

The trigger on each instance is set to a repetition interval of 1000ms. Whenever an instance is not able to finish processing this trigger within this repetition interval, an overflow occurs, i.e. the next trigger even fires before the previous one was completed.

For better visibility, only the master instance and 20 swarm instances were evaluated in detail. Randomly picked instances verified that the qualitative behavior was similar during the observed interval of time.

The test period started on 2015-04-21 at 10:30 UTC and ended 30 hours later on 2015-04-22 at 16:30 UTC. The most relevant observation at first glance is that the graphs may easily be divided into separate intervals of time: Some with relative steep increases in the number of overflows and some nearly flat intervals. When taken into account that the local time zone was UTC+2, the intervals with the steepest gradient were during normal office hours, approx. 08:30 to 17:15 local time.

## Conclusion

The test revealed that the gateway software runs stable, even if run in nearly 500 instances in parallel. The overall system load was relatively low; if broken down to a single instance (as the gateway would be used in real life applications), the system requirements are negligible. The main instance showed no obvious difference in behavior compared to the swarm instances and all swarm instances behave comparable to each other. The test was run on a “populated” network during normal office days. The distribution of network traffic over the time of day clearly reflected on the distribution of overflow events. It could be proven that the vast amount of publishes reaches the server. However, one should never rely on a 100% transmission rate, especially if a cell phone or radio connection to the internet is used, the network load is high or if the server is located far away. For time critical applications or safety related data, a local pre-processing or local buffering may be advised.

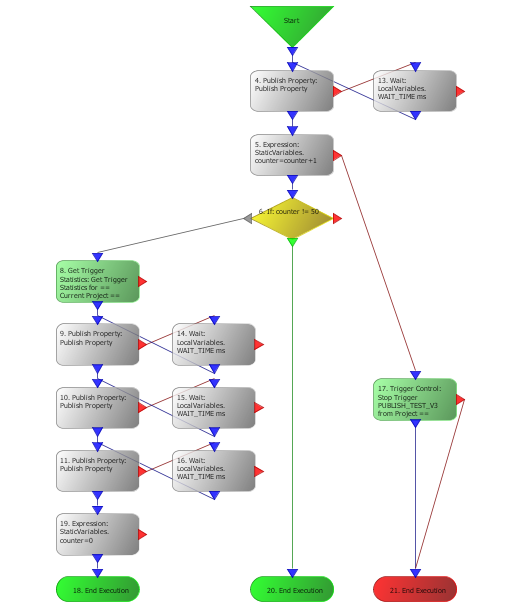
## Second test with modified trigger setup

The first long-term proved that the software ran stable and successfully transmitted most of the expected amount of data. However, due to network latencies and high traffic load, some transmit events took longer than the repetition interval which leads to overflows.

In order to not lose any transmission events, the triggers modified in a way more than one event may be executed at a time. The result was as predicted: no more overflows. The following graph shows the overflow and failure counters of the primary instance during the second test. Please note that the overflow counter was not reset before the test starts but it remains constant throughout the duration of the test. What is clearly visible in the graph is that at certain points in time, the failure rate jumps to a higher value and remains constant in-between. The timestamps where the jumps occurred could be matched perfectly with reported TR50 connections drops reported in the error log. The graphs and error times of randomly picked swarm instances were qualitatively the same as for the master instance so for clarity reasons the graph only shows the situation on instance #0.

## Third test with buffering trigger

For a third test, the trigger was enhanced further. Each trigger consists of up to four publish actions (depending on counter value). Whenever any of the publish property actions fails, an additional wait action was added and the transmission was retried after 5 seconds. The wait loops will be repeated as often as needed. Additionally, the triggers were allowed to run only once at a time (“Max in Progress” = 1) but with a queue length of 1000. That means that the trigger events for a time span of more than 15 minutes were able to queue up locally in case of a connection loss. When the connection becomes alive again, all queued events were processed in order of appearance.



The resulting failure and overflow counter graphs are quite boring: There were simply no overflows and failures at all during a 40 hour test. For this reason, the graph was omitted.

TODO: Results of lost events test