

**Deliverable D6.3**

**Assessment Report**

|  |  |
| --- | --- |
| Editor: | Paulo Chainho, Altice Labs |
| Deliverable nature: | (R) Document, Report |
| Dissemination level:(Confidentiality) | Public (PU) |
| Contractual delivery date: | 2016-06-30 |
| Actual delivery date: | 2016-06-30 |
| Suggested readers: | Project members and external users that use services deployed in the reTHINK Testbed |
| Version: | 1.0 |
| Total number of pages: | 39 |
| Keywords: | testbed, trials, demo scenarios, hyperties, assessment |

***Abstract***

This report is the first edition of a critical evaluation of the reTHINK framework performance according to phase 1 trials. This initial evaluation is more focused on technical information collected during the trials. Initial feedback from developers and hackathon participants were also considered.

[End of abstract]

Disclaimer

This document contains material, which is the copyright of certain reTHINK consortium parties, and may not be reproduced or copied without permission.

All reTHINK consortium parties have agreed to full publication of this document.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the reTHINK consortium as a whole, nor a certain part of the reTHINK consortium, warrant that the information contained in this document is capable of use, nor that use of the information is free from risk, accepting no liability for loss or damage suffered by any person using this information.

*This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 645342. This publication reflects only the author’s view and the European Commission is not responsible for any use that may be made of the information it contains.*

Impressum

Full project title: **Trustful hyper-linked entities in dynamic networks**

Short project title: **reTHINK**

Number and title of work-package: **WP6 – Testbeds, Integration and Validation**

Number and title of task: **Task 6.2: Testbed operation and demonstration activities;**

**Task 6.4: Trial Evaluations, Business and Architecture Assessment**

Document number and title: **D6.3Assessment Report**

Editor: **Paulo Chainho, Altice Labs**

Work-package leader: **Marc Emmelmann**, company: **Fraunhofer FOKUS**

Copyright notice

2016 Participants in project RETHINK

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit [http://creativecommons.org/licenses/by-nc-nd/3.0](http://creativecommons.org/licenses/by-nc-nd/3.0/)

Executive summary

This Report provides the first detailed assessment of reTHINK framework according to phase 1 implementation results. The current evaluation is more focused on technical information collected during ongoing phase 1 trials with preliminary feedback from scenario developers and participants to a mini-Hackathon. In general, the conducted assessment provides positive feedback and it is possible to conclude the project is progressing well and has already addressed some of its challenges, for example, service interoperability with minimum standardisation effort. Topics that require further work are:

* Framework and Service Deployment
* Identity Management
* Policy Management
* Discovery
* Quality of Service

The result of this evaluation provides an essential input to phase 2 activities.

List of authors

|  |  |  |
| --- | --- | --- |
| Company | Author | Contribution |
| PTIN | Paulo Chainho, Vitor Silva, Luis Duarte | General edition, Hyperty Runtime Evaluation |
| DT | Steffen Drüsedow | Technical evaluation of Message Nodes |
| Quobis | Antón Román Portabales | Developments Evaluations |
| INESC-ID | Ricardo Lopes Pereira, Rui Pereira | Domain Registry Evaluation |
| FOKUS | Marc Emmelmann | Catalogue Evaluation |
| ORANGE | Simon Bécot, Eric Paillet, Jean-Michel Crom | IDP Evaluation, Architecture Assessment |
| TU Berlin | Sebastian Goendoer | Global Registry Evaluation |

Table of Contents

[Executive summary 3](#_Toc455163664)

[List of authors 4](#_Toc455163665)

[Table of Contents 5](#_Toc455163666)

[List of Figures 7](#_Toc455163667)

[Abbreviations 8](#_Toc455163668)

[1 Introduction 9](#_Toc455163669)

[2 Technical Evaluations 10](#_Toc455163670)

[2.1 Message Node 10](#_Toc455163671)

[2.1.1 Description of Component 10](#_Toc455163672)

[2.1.2 Methodology and setup 10](#_Toc455163673)

[2.1.3 Conformance metrics 10](#_Toc455163674)

[2.1.4 Performance metrics 13](#_Toc455163675)

[2.1.5 Summary of MN Assessment 14](#_Toc455163676)

[2.2 Runtime 22](#_Toc455163677)

[2.2.1 Description of Component 22](#_Toc455163678)

[2.2.2 Metrics 22](#_Toc455163679)

[2.2.3 Tests 23](#_Toc455163680)

[2.3 Global Registry 27](#_Toc455163681)

[2.3.1 Description of component 27](#_Toc455163682)

[2.3.2 Metrics 27](#_Toc455163683)

[2.3.3 Tests 27](#_Toc455163684)

[2.4 Domain Registry 28](#_Toc455163685)

[2.4.1 Component Description 28](#_Toc455163686)

[2.4.2 Metrics 28](#_Toc455163687)

[2.4.3 Tests 29](#_Toc455163688)

[2.4.4 Future work 33](#_Toc455163689)

[2.5 Catalogue 34](#_Toc455163690)

[2.5.1 Description of Component 34](#_Toc455163691)

[2.5.2 Metrics 34](#_Toc455163692)

[2.5.3 Tests 35](#_Toc455163693)

[2.6 Identity Provider 43](#_Toc455163694)

[2.6.1 Description of Component 43](#_Toc455163695)

[2.6.2 Metrics 44](#_Toc455163696)

[2.6.3 Tests 44](#_Toc455163697)

[3 Development Evaluations 45](#_Toc455163698)

[3.1 ReTHINK Hyperty development evaluation survey 45](#_Toc455163699)

[3.2 ReTHINK Application development evaluation survey 45](#_Toc455163700)

[3.3 Results of reTHINK Hyerty development evaluation survey 46](#_Toc455163701)

[3.4 Results of reTHINK Application development evaluation survey 49](#_Toc455163702)

[4 Architecture Assessment 52](#_Toc455163703)

[4.1 General architecture requirements 52](#_Toc455163704)

[4.2 Interoperability 53](#_Toc455163705)

[4.3 Security and Identity Management 54](#_Toc455163706)

[4.4 QoS Management 55](#_Toc455163707)

[4.5 Multipary, Realtime services 55](#_Toc455163708)

[4.6 Deployment/exploitation 57](#_Toc455163709)

[5 References 59](#_Toc455163710)

[Annex A Message Node detailed tests results 60](#_Toc455163711)

[A.1 Vertx Tests 60](#_Toc455163712)

[A.1.1 Vertx conformance tests 60](#_Toc455163713)

[A.1.2 Vertx performance tests 62](#_Toc455163714)

[A.2 Matrix Tests 64](#_Toc455163715)

[A.2.1 Matrix conformance tests 64](#_Toc455163716)

[A.2.2 Matrix performance tests 66](#_Toc455163717)

[A.3 NodeJS Tests 68](#_Toc455163718)

[A.3.1 Nodejs conformance tests 68](#_Toc455163719)

[A.3.2 Nodejs performance tests 71](#_Toc455163720)

List of Figures

[Figure 1: Overview of specification conformance test results](file:///D:/Users/tassos/Documents/reTHINK/D6.3 Assessment Report_V1.docx" \l "_Toc455163721) 15

[Figure 2: Performance of Hyperty address allocations](file:///D:/Users/tassos/Documents/reTHINK/D6.3 Assessment Report_V1.docx" \l "_Toc455163722) 16

[Figure 3: Performance of Hyperty address de-allocations](file:///D:/Users/tassos/Documents/reTHINK/D6.3 Assessment Report_V1.docx" \l "_Toc455163723) 17

[Figure 4: Performance of Object address allocations](file:///D:/Users/tassos/Documents/reTHINK/D6.3 Assessment Report_V1.docx" \l "_Toc455163724) 18

[Figure 5: Performance of Object address de-allocations](file:///D:/Users/tassos/Documents/reTHINK/D6.3 Assessment Report_V1.docx" \l "_Toc455163725) 19

[Figure 6: Performance of Hyperty messaging](file:///D:/Users/tassos/Documents/reTHINK/D6.3 Assessment Report_V1.docx" \l "_Toc455163726) 20

[Figure 7: Performance for object update messages to N subscribers](file:///D:/Users/tassos/Documents/reTHINK/D6.3 Assessment Report_V1.docx" \l "_Toc455163727) 21

[Figure 8 – Hyperty Deployment in The Runtime 23](#_Toc455163728)

[Figure 9 – Protocol Stub deployed in the Runtime 24](#_Toc455163729)

[Figure 10 – IDP Proxy deployed in the Runtime 25](#_Toc455163730)

[Figure 11 – Hello World Reporter publishes updates to Hello World Data Object 26](#_Toc455163731)

[Figure 12 - Hello World Observer receives updates on Hello World Data Object 26](#_Toc455163732)

[Figure 13 – Global Registry average response time test 28](#_Toc455163733)

[Figure 14 - Deployment architecture 30](#_Toc455163734)

[Figure 15 - Test scenarios 30](#_Toc455163735)

[Figure 16 - Effective vs. solicited request rate 31](#_Toc455163736)

[Figure 17 - Average response time 32](#_Toc455163737)

[Figure 18 - Failure recovery test 33](#_Toc455163738)

[Figure 19 - System under test (SUT) for the Catalogue performance assessment 36](#_Toc455163739)

[Figure 20 - Illustration of a single httperf connection 37](#_Toc455163740)

[Figure 21 - Illustration of a multiple httperf connections 38](#_Toc455163741)

[Figure 22 - Catalogue Assessment: Response Time over Demanded Request Rate 40](#_Toc455163742)

[Figure 23 - Catalogue Assessment: Response Time over Demanded Request Rate (unsaturated system, showing 95% confidence values) 41](#_Toc455163743)

[Figure 24 - Catalogue Assessment: Illustates the the actual (i.e., measured) request rate over the demanded request rate 42](#_Toc455163744)

[Figure 25 - Catalogue Assessment: Reported errors 43](#_Toc455163745)

Abbreviations

|  |  |
| --- | --- |
| API | Application Programming Interface |
| CSP | Communication Service Provider |
| DHT | Distributed hash table |
| ECDSA | Elliptic Curve Digital Signature Algorithm |
| GUID | Global User Identifier |
| IdP | Identity Provider |
| JIT | Just In Time Compilation |
| JVM | Java Virtual Machine |
| MN | Message Node |
| ProtOfly | Protocol On-the Fly |
| QoS | Quality of Service |
| KPI | Key Performance Indicator |
| RTT | [Round-trip delay time](https://en.wikipedia.org/wiki/Round-trip_delay_time) |
| SUT | System Under Test |
| VM | Virtual Machine |

# Introduction

# Technical Evaluations

This chapter provides a detailed technical evaluation for each different sub-system implemented in phase 1 and deployed in the testbed, namely the Message Nodes (Vertx, NodeJS, NoMatrix), the Hyperty Runtime, the Domain Registry, the Global Registry, the Catalogue and the IDP.

The further development of the Matrix based Message Node has been frozen after phase 1. The phase 1 assessment results have shown that the Matrix architecture causes relatively long delays for the establishment of new communication channels, but is fast once the channel is established. Since the phase 1 reTHINK framework always generated new identifiers, it was decided to freeze and postpone the Matrix based developments until the persistence and re-use of identifiers was introduced to the core-framework. The core of the Matrix based MN implementation has been transferred to a new “NoMatrix” MN, which is now also the base of the DTAG testbed installations.

The NoMatrix MN is tested here together with the Vertx and Nodejs implementations. The original Matrix MN is not covered in these tests, since due to resource constraints it is not yet feature complete.

## Message Node

### Description of Component

The Message Node provides Real Time Message Oriented communication functionalities used by Hyperties to communicate with each other. It provides different communication patterns including publish/subscribe communication and Request/Response communication.

### Methodology and setup

For the evaluation of the individual Message Nodes (MNs) a set of Karma[10] tests has been developed. These common tests are executed against each Message Node implementation and the results are documented.

The set of tests has been updated to the latest specification changes and at some points slightly extended due to new features compared with the tests that have been described in the previous Deliverable D6.3.

The first set of tests evaluates the conformance of the MNs with the specified request/response patterns of the message API. The second set of tests evaluates the performance of the MNs in forced situations of increased load.

The individual tests are described in the following chapter.

In order to be comparable all tests have been performed on a single machine with locally deployed Message Nodes and a locally deployed Domain Registry. (This is the same setup that has been used for the tests in the previous Assessment report D6.3)

The used machine has following parameters:

* CPU: Intel Core i7-4712MQ, 2.30GHz (8 Cores)
* 8 GB RAM
* SSD
* OS: Debian (stretch)

The Domain Registry was running in its own Docker container in a custom Docker network, exposing the internal port 4567 to the host port 4567. So it was accessible from other docker containers in the same custom network as well as from the host via this port.

The individual MNs have been deployed from their pre-built docker-images available on the rethink docker-hub. All MNs have been operated as single nodes in the simpliest possible configuration. Potentially available cluster modes were not used.

### Conformance metrics

These tests evaluate the conformance of the MNs with the specified message patterns. Example requests are created according to the specification at [13] and sent to the Message Node. The corresponding responses are then compared with the expected results. Performance parameters, like the response time, are not evaluated in these tests. The following sub-sections give a short description of the individual conformance tests.

#### Stub connection and disconnection

The purpose of this test is to ensure that the Stub exposes the "status" events as specified in the Protocolstub state machine at <https://github.com/reTHINK-project/specs/blob/master/messaging-framework/protostub-state-machine.png>.

It instantiates a stub and performs a connection and then a disconnection of the stub to the corresponding MN. The tests check that the stubs are generating the correct status event sequence (“created”, “in-progress”, “live”, “disconnected”) for each connection/disconnection situation.

|  |  |
| --- | --- |
| Test 1.1 | Stub connect/disconnect |

#### Hyperty address allocation messages

The purpose of this test is to ensure the Conformance of the MN operations with the message specification for hyperty address allocationshttps://github.com/reTHINK-project/dev-service-framework/blob/master/docs/specs/messages/address-allocation-messages.md.

It checks the allocation and de-allocation of hyperty addresses by the MNs. It includes following sub-tests:

|  |  |
| --- | --- |
| Test 2.1 | allocation of a single address |
| Test 2.2 | de-allocation of a single address |
| Test 2.3 | allocation of an address block of 3 addresses |
| Test 2.4 | de-allocation of an address block of 3 addresses by given address array |
| Test 2.5 | allocation of an address block with a given “allocationKey” as identifier for this block of addresses |
| Test 2.6 | de-allocation of an address block identified by a given allocationKey |

#### Object address allocation messages

The purpose of this test is to ensure the Conformance of the MN operations with the message specification for object address allocations.

It checks the allocation and de-allocation of object addresses by the MNs. It includes following sub-tests:

|  |  |
| --- | --- |
| Test 3.1 | allocation of a single address incl. check that a given "scheme" is used for the object address allocations |
| Test 3.2 | de-allocation of a single address |
| Test 3.3 | allocation of an address block of 3 addresses |
| Test 3.4 | de-allocation of an address block of 3 addresses by given address array |
| Test 3.5 | allocation of an address block with a given “allocationKey” as identifier for this block of addresses |
| Test 2.6 | de-allocation of an address block identified by a given “allocationKey” |

#### Subscription and object update messages

The purpose of this test is to ensure the Conformance of the MN operations with the Data synchronization message specification.

It checks the subscription and un-subscription for given object addresses at the MN as well as the correct publication of object update events to the subscribers. It includes following sub-tests:

|  |  |
| --- | --- |
| Test 4.0 | Test setup: allocation of an object address |
| Test 4.1 | subscription for an object address with a given body.source attribute (the given body.source must be used as subscriber url) |
| Test 4.2 | subscription for an object address without a body.source attribute (the from address of the subscribe message must be used as subscriber url) |
| Test 4.4 | update of the subscribed object by the reporter and expect correct update events on both subscribers |
| Test 4.4 | unsubscription of both subscribers, expecting correct 200 OK response |
| Test 4.5 | another update of the subscribed object and expect that no events are received by the (now unsubscribed) subscribers |

#### Registration messages

The purpose of this test is to ensure the Conformance of the MN operations with the Registration message Specification [11].

This set of tests requires a running Domain Registry component that is accessible by the MNs. The MN uses a RegistryConnector component to forward the received messages to the Domain Registry and receive back the responses.

It includes following sub-tests:

|  |  |
| --- | --- |
| Test 5.1 | Test setup: registration of an allocated hyperty address for a given userid |
| Test 5.2 | Search a hyperty by given userid |
| Test 5.3 | Search a hyperty address by given userid and scheme |
| Test 5.4 | Search a hyperty by given userid and resource-types |
| Test 5.5 | Search a hyperty by given userid, scheme and resource-types |
| Test 5.6 | Search and retrieve hyperty-data by a given hyperty address |
| Test 5.7 | Search for a non existing registry object/hyperty → expect correct error message |
| Test 5.8 | Test correct handling of keep-alive messages for an active registration |
| Test 5.9 | Subscription for update events, when the status of the registered object changes |
| Test 5.10 | Test delivery and receiving of status update events (very NEW in spec.) |
| Test 5.11 | unregistration of a hyperty address |

### Performance metrics

These tests evaluate the performance and robustness of the MNs in forced situations of increased load. Therefore requests are sent in loops of increasing iteration counts and with differing message sizes as fast as possible. For each of these tests, the duration is measured and used as metric for the evaluation.

Each test is executed 3 times. The results in the table are the mean values of the 3 test runs. Messages are sent out in iterations as fast as possible without any delays between messages to put the MN under stress.

#### Hyperty address allocation messages

These tests create and send an increasing number of allocation and de-allocation messages for Hyperty addresses to the MNs and expect at each time a correct 200 OK response. The sizes of these messages are already defined by the specification. They can not be smaller than specified and it also makes no sense to artificially increase their size. Therefore only the number of iterations and the number of addresses to allocate vary. Furthermore the tests with several address numbers are executed with and without an “allocationKey” parameter, which can be used to identify blocks of allocated addresses later on (e.g. for an de-allocation).

Also the time for the de-allocation of all addresses is measured.

Following measurements are performed in this set of test:

* hyperty address allocation and de-allocation requests for 1 address each for 100, 1000 and 10000 iterations
* hyperty address allocation and de-allocation requests for a block of 3 address each without an “allocationKey” for 100, 1000 and 10000 iterations
* hyperty address allocation and de-allocation requests for a block of 3 address each with “allocationKey” for 100, 1000 and 10000 iterations

#### Object address allocation messages

These tests repeat the same procedure as previously described for hyperty addresses now for object addresses too. The same comments and notes apply here as well.

Following measurements are performed in this set of tests:

* Object address allocation and de-allocation requests for 1 address each for 100, 1000 and 10000 iterations
* Object address allocation and de-allocation requests for a block of 3 address each without “allocationKey” for 100, 1000 and 10000 iterations
* Object address allocation and de-allocation requests for a block of 3 address each with “allocationKey” for 100, 1000 and 10000 iterations

#### Hyperty messages

These tests send an increasing number of messages from one allocated Hyperty to a second allocated Hyperty. Each Hyperty is connected via its own stub. This simulates the behaviour of messaging between Hyperties which are deployed in two different Runtimes.

During the tests the payload sizes are increased from 100B over 1kB to 10kB. The purpose of this test is to ensure that all messages arrive in the same order they were sent and to measure the overall needed time.

The measured time frame starts when the first message from Hyperty 1 via Stub 1 is sent and stops when the last message is received by Hyperty 2 via Stub 2.

Following measurements are performed:

* Message with payload of 100B for 100, 1000 and 10000 iterations
* Message with payload of 1kB for 100, 1000 and 10000 iterations
* Message with payload of 10kB for 100, 1000 and 10000 iterations

#### Subscription and Publication of Object updates

These tests send a publish Object update message from one reporter to a increasing number of subscribers. During the tests also the payload sizes are increased from 100B over 1kB to 10kB.

For this purpose a single reporter Hyperty is connected to the MN via a first stub and an increasing number of subscriber hyperties are connected via their own stubs, one stub for each subscriber hyperty. In the preparation phase, the reporter stub allocates an object address first and then all subscriber Hyperties send subscription messages for updates on this object and wait for their OK response.

The measured time frame starts when the Reporter Hyperty sends the update message and stops when all subscribers have received the corresponding update message. The order in which the subscribers receive the update message is not considered as test criteria.

Following measurements are performed:

* Update Message with payload of 100B for 100, 200, and 1000 subscribers
* Update Message with payload of 1kB for 100, 200, and 1000 subscribers
* Update Message with payload of 10kB for 100, 200, and 1000 subscribers

### Summary of MN Assessment

#### Conformance test results

The following figure summarizes the results of the conformance tests for all 3 Message Nodes.

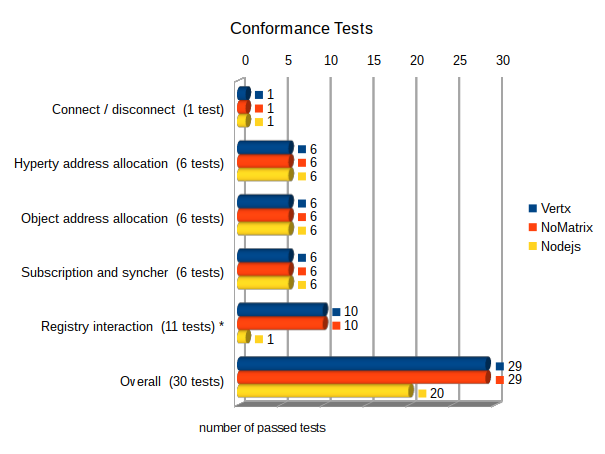


Figure 1: Overview of specification conformance test results

***Connection and status event emission test***

All 3 nodes pass the basic connect and disconnect test and emitted the correct sequence of status events during the deployment and connection phase.

***Address allocation / de-allocation tests:***

Also all tests for the allocation and de-allocation of hyperty and object addresses where passed successfully by all 3 message nodes.

***Subscription and publish:***

All 3 nodes have passed the test for subscription and publication of data object updates according to the latest specification

***Registry interactions:***

**Notes:**

For the interaction with the Domain Registry all 3 nodes make use of a RegistryConnector component that is available as a plug-in module. Therefore the pure message format and exchange between the MN and the domain registry is identical for all 3 MNs. Differences are only be caused by the way how messages are identified and assigned to the RegistryConnector plugin and how responses are parsed and routed back.

All MNs are failing to pass Test 5.10 (Test delivery and receiving of status update events). This test checks a quite recent addition of the spec. which allows any runtime to subscribe for status-changes of hyperties or objects in the domain registry. This functionality will require an architectural extension of the RegistryConnector, because it needs to implement a server interface as well in order to receive notification from the domain registry. So far it only acts in the client role, sending requests and to the domain registry and expecting responses. This extension is – at the time of test execution - not yet implemented in the RegistryConnector and therefore the corresponding test fails for all nodes using it.

The mentioned new feature is not yet part of the rethink core framework – therefore this failed test is not crucial for the use of the MNs.

**Results:**

The Vertx and the NoMatrix node are passing all registry interaction tests successfully, while the NodeJS node is failing on 10 of 11 tests. This seems to be caused by a relatively recent change in the RegistryConnector in the way how the JSON responses from the domain registry are parsed and handed back to the MN implementation. The necessary adaptations have not been applied to the NodeJS node, which caused the tests to fail. This issue has been reported to the component owner and the work on a fix was ongoing but not yet available at the time of test execution.

#### Performance test results

***Allocation and de-allocation of Hyperty addresses***

The following 2 figures provide a summary of the performance of the MNs in terms of allocation and de-allocation of hyperty addresses.

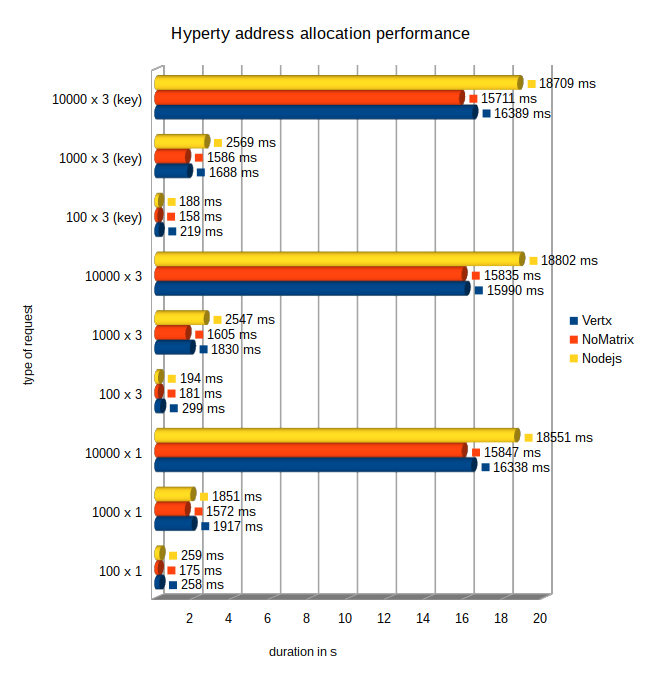


Figure 2: Performance of Hyperty address allocations

All 3 nodes successfully passed all hyperty address allocation tests. The performance of the hyperty allocations is on a comparable level for all nodes, while the NoMatrix MN performs slightly faster than the Vertx and Nodejs MNs. The NodeJS MN became a little bit slower especially for higher iteration numbers and for tests with several address allocations at once.

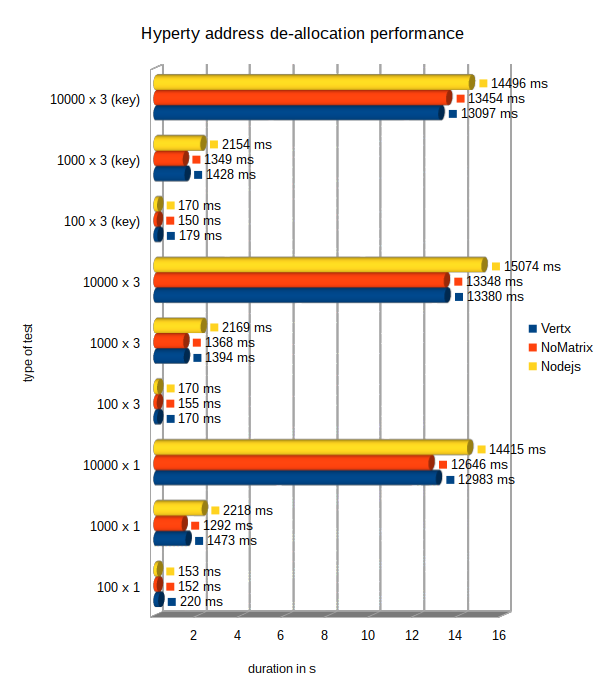


Figure 3: Performance of Hyperty address de-allocations

All 3 nodes passed successfully all hyperty address de-allocation tests. The performance of the de-allocations is in general up to 20% better than for the hyperty address allocations.

The NoMatrix MN performs slightly faster than the Vertx and Nodejs MNs. The NodeJS MN became a little bit slower especially for higher iteration numbers and for tests with several address de-allocations at once.

***Allocation and de-allocation of Object addresses***

The following 2 figures provide a summary of the performance of the MNs in terms of allocation and de-allocation of Object addresses.

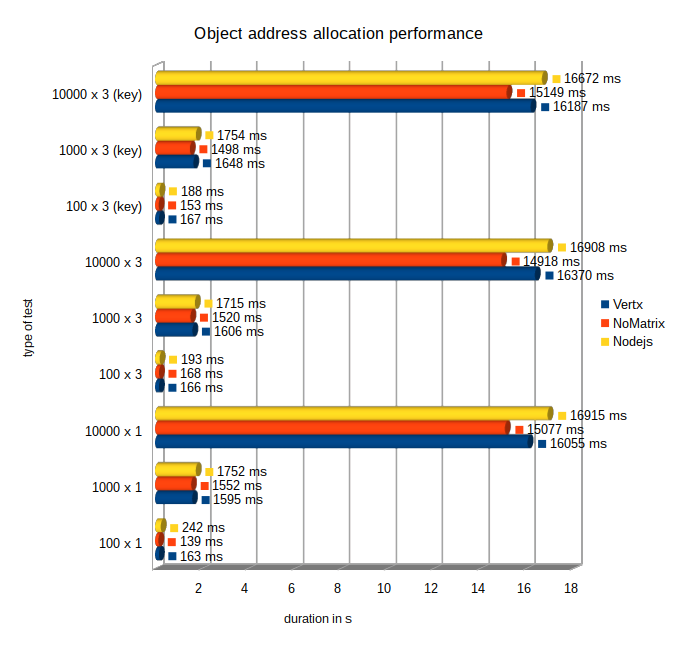


Figure 4: Performance of Object address allocations

All 3 nodes passed successfully all object address allocation tests. The NoMatrix MN performs slightly faster than the Vertx, which again is slightly faster than the Nodejs MNs. But in general the performance is on comparable level.

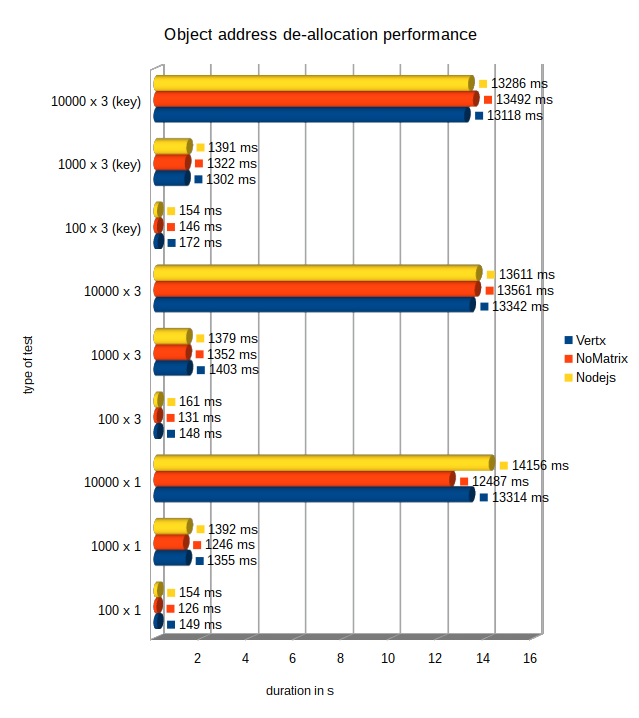


Figure 5: Performance of Object address de-allocations

All 3 nodes passed successfully all object address de-allocation tests. Also for the object addresses the performance of the de-allocations is in general up to 20% better than for the allocations.

The performance of the 3 nodes is on a comparable level, again with a small advantage for the NoMatrix MN in most of the tests.

***Hyperty messaging performance***

The following figure summarizes the results of the performance for the sending of message with different sizes from one Hyperty to another.

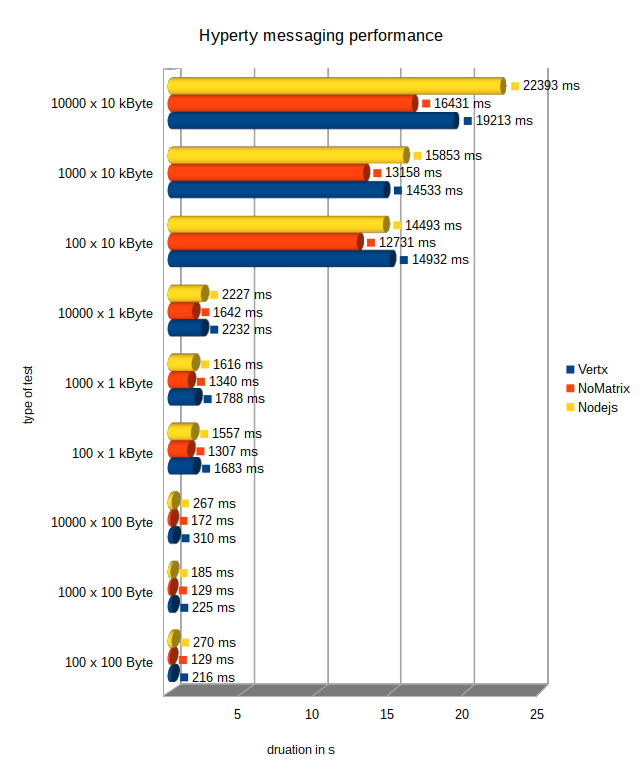


Figure 6: Performance of Hyperty messaging

All 3 nodes have passed successfully all hyperty messaging performance tests.

The NoMatrix MN shows the best performance in all different runs while the NodeJS and Vertx node are on nearly the same level.

***Object status update performance***

The next figure shows the results of the performance measurements for the delivery of object update messages to a variable number of subscribers.

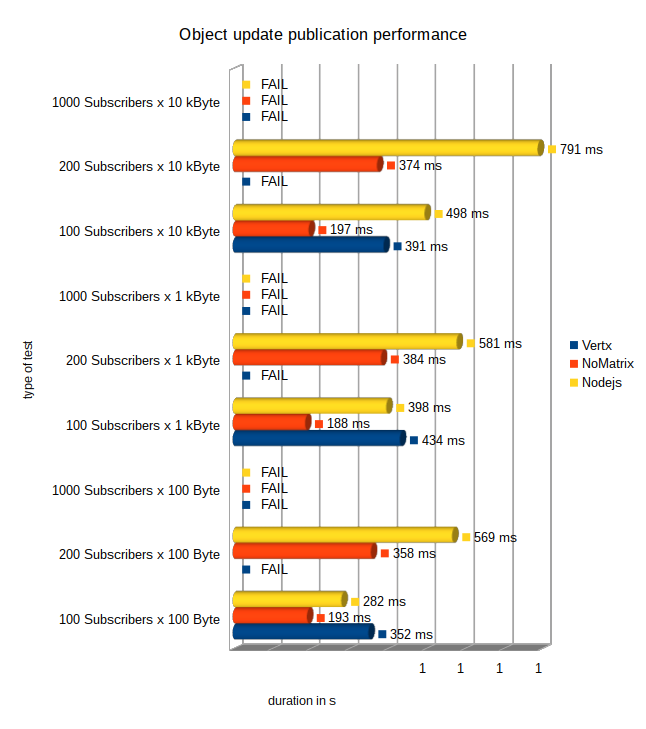


Figure 7: Performance for object update messages to N subscribers

**Note:**

All tests with 1000 subscribers for a event were expected to fail for all 3 nodes. The reason is that the tests were performed using the karma testscripts inside of a google-chrome browser. Since the common browsers have resource limitations regarding the parallel established of outgoing connections, the tests with 1000 subscribers were inhibited. The empirical limit for parallel connections was found around 250.

The NoMatrix and the NodeJS MNs have passed all tests with 100 and 200 subscribers and, as expected, failed for 1000 parallel subscribers due to the mentioned browser limitations. The NoMatrix node performed conspicuously faster than the NodeJS node.

The Vertx node has passed the tests with 100 but not with 200 subscribers. The reason for the failure was that not all subscribers received an event notification when an object update was performed. Variying numbers between 4 and 9 events got lost. Since the tests count only successful when all subscribers receive the update notifications the test failed. The issue has been reported and the reason for these lost events was still under investigation by the component owner at the time of the test documentation.

#### MN Assessment conclusions and recommendations

The tests have shown that the NoMatrix MN can be recommended as the “all purpose” MN, passing all mandatory conformance- and performance- tests and providing the best performance in most of the cases.

The Vertx MN is passing all conformance- and most of the performance- tests. It is the standard MN for most of the applied testbeds in reTHINK and has proven its reliability in the daily work. The lost events for very high numbers of subscribers for updates on a data object seem to have a rather low relevance in the real life operation, according to the practical experiences. Nevertheless they indicate a potential problem in the routing engine that must be investigated.

The NodeJS MN is showing very good performance results but it is failing in the tests for the domain-registry interaction, which are crucial for the use of the MN in a testbed. These failures can be fixed by the proper integration of the latest RegistryConnector version. If this is done the NodeJS MN is a good option for the operation of a reTHINK testbed.

# References

1. Deliverable D2.1, “Framework Architecture Definition”, 31-07-2015.
2. Deliverable D2.2, “Data Models and Interface Specification of the Framework ”, 30-08-2015
3. Deliverable D3.1, “Hyperty-Runtime-and-Hyperty-Messaging-Node-Specification”, 30-09-2015.
4. Deliverable D4.1, “Management and Security features specifications”, 30-09-2015.
5. Deliverable D3.2, “Hyperty Runtime and Hyperty Message Node Phase 1”, 29-02-2016.
6. Deliverable D4.2, “Implementation of Governance and identity management components for phase 1”, 29-02-2016.
7. Deliverable D5.1, “Service Scenarios Specification”, 29-12-2015.
8. Deliverable D5.2, “Initial Hyperties and Service Scenarios implementation”, 29-06-2016.
9. Deliverable D6.1, “Initial Hyperties and Service Scenarios implementation”, 29-06-2016.
10. Karma testing tool, https://karma-runner.github.io/1.0/index.html; last accessed June 2016.
11. Registration Message specification, <https://github.com/reTHINK-project/dev-service-framework/blob/master/docs/specs/messages/registration-messages.md>; last accessed June 2016.
12. Hello World Hyperty, <https://github.com/reTHINK-project/dev-hyperty/tree/master/src/hello-world>; last accessed June 2016.
13. Messages Specification: <https://github.com/reTHINK-project/dev-service-framework/tree/master/docs/specs/messages>; last accessed June 2016.
14. Autobench GitHub Repository. Available online: <https://github.com/menavaur/Autobench>; last accessed June 2016.
15. Httperf GitHub Repository. Available online: <https://github.com/httperf/httperf>; last accessed June 2016.
16. Httperf Man Page. Available online: <http://www.labs.hpe.com/research/linux/httperf/httperf-man-0.9.ps>; last accessed June 2016.
17. WebRTC Security architecture, <https://tools.ietf.org/html/draft-ietf-rtcweb-security-arch-12>; last accessed June 2016.
18. [Domain Registry user manual](https://github.com/reTHINK-project/dev-registry-domain/blob/master/docs/DomainRegistryUserManual.md), <https://github.com/reTHINK-project/dev-registry-domain/blob/master/docs/DomainRegistryUserManual.md>; last accessed June 2016.
19. [HAProxy](http://www.haproxy.org/) GitHub Repository. Available online: http://www.haproxy.org/; last accessed June 2016.
20. Apache Cassandra site. Available online: http://cassandra.apache.org/; last accessed June 2016.
21. agmoyano/OpenIDConnect GitHub Repository. Available online: <https://github.com/agmoyano/OpenIDConnect>; last accessed June 2016.
22. PhPOIDC Bitbucket Repository. Available online: <https://bitbucket.org/PEOFIAMP/phpoidc> ; last accessed June 2016.
23. Message Node detailed tests results
    1. Vertx Tests

The Vertx MN was deployed as docker container on the test machine according to the instructions at the Vertx MN github page.

* + 1. Vertx conformance tests

**1. Stub Connection and Disconnection**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Test** | **Result** | **Comments** |
| 1.1 | connect / disconnect | **OK** |  |

**2. Hyperty address allocation messages**

| **No.** | **Test** | **Result** | **Comments** |
| --- | --- | --- | --- |
| 2.1 | allocate single hyperty address | **OK** |  |
| 2.2 | de-allocate single hyperty address | **OK** |  |
| 2.3 | allocate block of hyperty addresses | **OK** |  |
| 2.4 | de-allocate block of hyperty addresses | **OK** |  |
| 2.5 | allocate block of hyperty addresses with allocationKey | **OK** |  |
| 2.6 | de-allocate block of hyperty addresses by allocationKey | **OK** |  |

**3. Object address allocation messages**

| **No.** | **Test** | **Result** | **Comments** |
| --- | --- | --- | --- |
| 3.1 | Allocate single object address | **OK** |  |
| 3.2 | de-allocate single object address | **OK** |  |
| 3.3 | allocate block of object addresses | **OK** |  |
| 3.4 | de-allocate block of object addresses | **OK** |  |
| 3.5 | allocate block of object addresses with allocationKey | **OK** |  |
| 3.6 | de-allocate block of object addresses by allocationKey | **OK** |  |

**4. Subscription and publish messages**

| **No.** | **Test** | **Result** | **Comments** |
| --- | --- | --- | --- |
| 4.1 | Allocate object address | **OK** |  |
| 4.2 | subscribe for object address with body.source | **OK** |  |
| 4.3 | subscribe for object address without body.source | **OK** |  |
| 4.4 | publish attribute update and receive update on both subscribers | **OK** |  |
| 4.5 | unsubscribe both subscribers | **OK** |  |
| 4.6 | publish second attribute update and don’t update on both subscribers | **OK** |  |

**5. Hyperty registration messages**

| **No.** | **Test** | **Result** | **Comments** |
| --- | --- | --- | --- |
| 5.1 | Test setup: registration of an allocated hyperty address for a given userid | **OK** |  |
| 5.2 | Search a hyperty by given userid | **OK** |  |
| 5.3 | Search a hyperty address by given userid and scheme | **OK** |  |
| 5.4 | Search a hyperty by given userid and resource-types | **OK** |  |
| 5.5 | Search a hyperty by given userid, scheme and resource-types | **OK** |  |
| 5.6 | Search and retrieve hyperty-data by a given hyperty address | **OK** |  |
| 5.7 | Search for a non existing registry object/hyperty → expect correct error message | **OK** |  |
| 5.8 | Test correct handling of keep-alive messages for an active registration | **OK** |  |
| 5.9 | Subscription for update events, when the status of the registered object changes | **OK** |  |
| 5.10 | Test delivery and receiving of status update events | **FAIL** | Expected behavior,  Not implemented by RegistryConnector |
| 5.11 | unregistration of a hyperty address | **OK** |  |

* + 1. Vertx performance tests

**1. Allocate /de-allocate Hyperty addresses**

NOTE: Since the de-allocation of hyperty addresses was not implemented yet by the Vertx MN, the de-allocation tests have been commented out for now to let the overall test-suite proceed.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No** | **Test** | **No. of addresses per iteration** | **With alloc. Key ?** | **Duration in milli-seconds for different no. of iterations** | | | **Comment** | |
| **100** | **1k** | **10k** | |  |
| 1.1 | Allocate | 1 |  | 258 | 1917 | 16338 | |  |
| 1.2 | De-allocate | 1 |  | 220 | 1473 | 12983 | |  |
| 1.3 | Allocate | 3 |  | 299 | 2830 | 15990 | |  |
| 1.4 | De-allocate | 3 |  | 170 | 2394 | 13380 | |  |
| 1.5 | Allocate | 3 | x | 219 | 1688 | 16389 | |  |
| 1.6 | De-allocate | 3 | x | 179 | 1428 | 13097 | |  |

**2. Allocate /de-allocate Object addresses**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No** | **Test** | **No. of addresses per iteration** | **With alloc. Key ?** | **Duration in milli-seconds for different no. of iterations** | | | **Comment** | |
| **100** | **1k** | **10k** | |  |
| 2.1 | Allocate | 1 |  | 163 | 1595 | 16055 | |  |
| 2.2 | De-allocate | 1 |  | 149 | 135 | 13314 | |  |
| 2.3 | Allocate | 3 |  | 166 | 1606 | 16370 | |  |
| 2.4 | De-allocate | 3 |  | 148 | 1403 | 13342 | |  |
| 2.5 | Allocate | 3 | x | 167 | 1648 | 16187 | |  |
| 2.6 | De-allocate | 3 | x | 172 | 1302 | 13118 | |  |

**3. Hyperty messaging**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No** | **Test** | **Duration in milli-seconds for different no. of iterations** | | | **Comment** | |
| **100** | **1000** | **10000** | |  |
| 3.1 | 100 Byte payload | 216 | 1683 | 14932 | |  |
| 3.2 | 1kByte payload | 225 | 1788 | 14533 | |  |
| 3.3 | 10kByte payload | 310 | 2232 | 19213 | |  |

**4. Subscription and Object update publication**

**NOTE:**

The tested VertX MN did not always forward the published events to all subscribed listeners. Especially in the runs with 200 listeners variying numbers of subscribers (2 to 9) did not receive the event. Since the completeness was a test criteria, the whole test failed in such a case.

The results for this test were constrained by the limited Websocket resources in browser. This is the reason why the runs with 1000 iterations where expected to fail.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No** | **Test** | **Duration in milli-seconds for different no. of subscribers** | | | **Comment** | |
| **100** | **200** | **1000** | |  |
| 4.1 | 100 Byte payload | 352 | **FAIL** | **FAIL** | | 5 Lost Events in 200 iterations run;  Failure with 1000 iterations due to Limited Websocket resources in browser |
| 4.2 | 1kByte payload | 434 | **FAIL** | **FAIL** | | 2-5 Lost Events in 200 iterations run;  Failure with 1000 iterations due to Limited Websocket resources in browser |
| 4.3 | 10kByte payload | 391 | **FAIL** | **FAIL** | | 3-5 Lost Events in 200 iterations run;  Failure with 1000 iterations due to Limited Websocket resources in browser |

* 1. NoMatrix Tests

The NoMatrix MN was deployed and configured as a docker container according to the descriptions at <https://github.com/reTHINK-project/dev-msg-node-nomatrix>.

* + 1. NoMatrix conformance tests

**1. Stub Connection and Disconnection**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Test** | **Result** | **Comments** |
| 1.1 | connect / disconnect | **OK** |  |

**2. Hyperty address allocation messages**

| **No.** | **Test** | **Result** | **Comments** |
| --- | --- | --- | --- |
| 2.1 | allocate single hyperty address | **OK** |  |
| 2.2 | de-allocate single hyperty address | **OK** |  |
| 2.3 | allocate block of hyperty addresses | **OK** |  |
| 2.4 | de-allocate block of hyperty addresses | **OK** |  |
| 2.5 | allocate block of hyperty addresses with allocationKey | **OK** |  |
| 2.6 | de-allocate block of hyperty addresses by allocationKey | **OK** |  |

**3. Object address allocation messages**

| **No.** | **Test** | **Result** | **Comments** |
| --- | --- | --- | --- |
| 3.1 | Allocate single object address | **OK** |  |
| 3.2 | de-allocate single object address | **OK** |  |
| 3.3 | allocate block of object addresses | **OK** |  |
| 3.4 | de-allocate block of object addresses | **OK** |  |
| 3.5 | allocate block of object addresses with allocationKey | **OK** |  |
| 3.6 | de-allocate block of object addresses by allocationKey | **OK** |  |

**4. Subscription and publish messages**

| **No.** | **Test** | **Result** | **Comments** |
| --- | --- | --- | --- |
| 4.1 | Allocate object address | **OK** |  |
| 4.2 | subscribe for object address with body.source | **OK** |  |
| 4.3 | subscribe for object address without body.source | **OK** |  |
| 4.4 | publish attribute update and receive update on both subscribers | **OK** |  |
| 4.5 | unsubscribe both subscribers | **OK** |  |
| 4.6 | publish second attribute update and don’t update on both subscribers | **OK** |  |

**5. Hyperty registration messages**

| **No.** | **Test** | **Result** | **Comments** |
| --- | --- | --- | --- |
| 5.1 | Test setup: registration of an allocated hyperty address for a given userid | **OK** |  |
| 5.2 | Search a hyperty by given userid | **OK** |  |
| 5.3 | Search a hyperty address by given userid and scheme | **OK** |  |
| 5.4 | Search a hyperty by given userid and resource-types | **OK** |  |
| 5.5 | Search a hyperty by given userid, scheme and resource-types | **OK** |  |
| 5.6 | Search and retrieve hyperty-data by a given hyperty address | **OK** |  |
| 5.7 | Search for a non existing registry object/hyperty → expect correct error message | **OK** |  |
| 5.8 | Test correct handling of keep-alive messages for an active registration | **OK** |  |
| 5.9 | Subscription for update events, when the status of the registered object changes | **OK** |  |
| 5.10 | Test delivery and receiving of status update events | **FAIL** | Expected behavior,  Not implemented in RegistryConnector |
| 5.11 | unregistration of a hyperty address | **OK** |  |

* + 1. NoMatrix performance tests

**1. Allocate /de-allocate Hyperty addresses**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No** | **Test** | **No. of addresses per iteration** | **With alloc. Key ?** | **Duration in milli-seconds for different no. of iterations** | | | **Comment** | |
| **100** | **1k** | **10k** | |  |
| 1.1 | Allocate | 1 |  | 175 | 1572 | 15847 | |  |
| 1.2 | De-allocate | 1 |  | 152 | 1292 | 12646 | |  |
| 1.3 | Allocate | 3 |  | 181 | 160 | 15835 | |  |
| 1.4 | De-allocate | 3 |  | 155 | 1368 | 13348 | |  |
| 1.5 | Allocate | 3 | x | 158 | 1586 | 15711 | |  |
| 1.6 | De-allocate | 3 | x | 150 | 1349 | 13454 | |  |

**2. Allocate /de-allocate Object addresses**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No** | **Test** | **No. of addresses per iteration** | **With alloc. Key ?** | **Duration in milli-seconds for different no. of iterations** | | | **Comment** | |
| **100** | **1k** | **10k** | |  |
| 2.1 | Allocate | 1 |  | 139 | 1552 | 15077 | |  |
| 2.2 | De-allocate | 1 |  | 126 | 1246 | 12487 | |  |
| 2.3 | Allocate | 3 |  | 168 | 1520 | 14918 | |  |
| 2.4 | De-allocate | 3 |  | 131 | 1352 | 13561 | |  |
| 2.5 | Allocate | 3 | x | 153 | 1498 | 15149 | |  |
| 2.6 | De-allocate | 3 | x | 146 | 1322 | 13492 | |  |

**3. Hyperty messaging**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No** | **Test** | **Duration in milli-seconds for different no. of iterations** | | | **Comment** | |
| **100** | **1000** | **10000** | |  |
| 3.1 | 100 Byte payload | 129 | 1307 | 12731 | |  |
| 3.2 | 1kByte payload | 129 | 1340 | 13158 | |  |
| 3.3 | 10kByte payload | 172 | 1642 | 16431 | |  |

**4. Subscription and Object update publication**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No** | **Test** | **Duration in milli-seconds for different no. of subscribers** | | | **Comment** | |
| **100** | **200** | **1000** | |  |
| 4.1 | 100 Byte payload | 193 | 358 | **FAIL** | | Expected, Limited Websocket resources in browser |
| 4.2 | 1kByte payload | 188 | 384 | **FAIL** | | Limited Websocket resources in browser |
| 4.3 | 10kByte payload | 197 | 374 | **FAIL** | | Limited Websocket resources in browser |

* 1. NodeJS Tests

The NodeJS MN was deployed and configured as a docker container according to the descriptions at <https://github.com/reTHINK-project/dev-msg-node-nodejs>. As described there also a required “redis” docker has been installed and setup for these tests.

* + 1. Nodejs conformance tests

**1. Stub Connection and Disconnection**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Test** | **Result** | **Comments** |
| 1.1 | connect / disconnect | **OK** |  |

**2. Hyperty address allocation messages**

| **No.** | **Test** | **Result** | **Comments** |
| --- | --- | --- | --- |
| 2.1 | allocate single hyperty address | **OK** |  |
| 2.2 | de-allocate single hyperty address | **OK** |  |
| 2.3 | allocate block of hyperty addresses | **OK** |  |
| 2.4 | de-allocate block of hyperty addresses | **OK** |  |
| 2.5 | allocate block of hyperty addresses with allocationKey | **OK** |  |
| 2.6 | de-allocate block of hyperty addresses by allocationKey | **OK** |  |

**3. Object address allocation messages**

| **No.** | **Test** | **Result** | **Comments** |
| --- | --- | --- | --- |
| 3.1 | Allocate single object address | **OK** |  |
| 3.2 | de-allocate single object address | **OK** |  |
| 3.3 | allocate block of object addresses | **OK** |  |
| 3.4 | de-allocate block of object addresses | **OK** |  |
| 3.5 | allocate block of object addresses with allocationKey | **OK** |  |
| 3.6 | de-allocate block of object addresses by allocationKey | **OK** |  |

**4. Subscription and publish messages**

| **No.** | **Test** | **Result** | **Comments** |
| --- | --- | --- | --- |
| 4.1 | Allocate object address | **OK** |  |
| 4.2 | subscribe for object address with body.source | **OK** |  |
| 4.3 | subscribe for object address without body.source | **OK** |  |
| 4.4 | publish attribute update and receive update on both subscribers | **OK** |  |
| 4.5 | unsubscribe both subscribers | **OK** |  |
| 4.6 | publish second attribute update and don’t update on both subscribers | **OK** |  |

**5. Hyperty registration messages**

**NOTE:**

The NodeJS MN seems to use an outdated version of the RegistryConnector, which causes errors during the parsing process of messages. During the tests this did lead to crashes of the NodeJS MN and therefore all related tests failed.

| **No.** | **Test** | **Result** | **Comments** |
| --- | --- | --- | --- |
| 5.1 | Test setup: registration of an allocated hyperty address for a given userid | **OK** |  |
| 5.2 | Search a hyperty by given userid | **FAIL** | Not updated to latest spec. |
| 5.3 | Search a hyperty address by given userid and scheme | **FAIL** | See above |
| 5.4 | Search a hyperty by given userid and resource-types | **FAIL** | See above |
| 5.5 | Search a hyperty by given userid, scheme and resource-types | **FAIL** | See above |
| 5.6 | Search and retrieve hyperty-data by a given hyperty address | **FAIL** | See above |
| 5.7 | Search for a non existing registry object/hyperty → expect correct error message | **FAIL** | See above |
| 5.8 | Test correct handling of keep-alive messages for an active registration | **FAIL** | See above |
| 5.9 | Subscription for update events, when the status of the registered object changes | **FAIL** | See above |
| 5.10 | Test delivery and receiving of status update events | **FAIL** | See above |
| 5.11 | unregistration of a hyperty address | **FAIL** | See above |

* + 1. Nodejs performance tests

**Note:**

The address de-allocation tests have been disabled for the NodeJS MN, because they are not implemented and would cause the overall test to fail.

**1. Allocate /de-allocate Hyperty addresses**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No** | **Test** | **No. of addresses per iteration** | **With alloc. Key ?** | **Duration in milli-seconds for different no. of iterations** | | | **Comment** | |
| **100** | **1k** | **10k** | |  |
| 1.1 | Allocate | 1 |  | 259 | 1851 | 18551 | |  |
| 1.2 | De-allocate | 1 |  | 153 | 2218 | 14415 | |  |
| 1.3 | Allocate | 3 |  | 194 | 2547 | 18802 | |  |
| 1.4 | De-allocate | 3 |  | 170 | 2169 | 15074 | |  |
| 1.5 | Allocate | 3 | x | 188 | 2569 | 18709 | |  |
| 1.6 | De-allocate | 3 | x | 170 | 2154 | 14496 | |  |

**2. Allocate /de-allocate Object addresses**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No** | **Test** | **No. of addresses per iteration** | **With alloc. Key ?** | **Duration in milli-seconds for different no. of iterations** | | | **Comment** | |
| **100** | **1k** | **10k** | |  |
| 2.1 | Allocate | 1 |  | 242 | 1752 | 16915 | |  |
| 2.2 | De-allocate | 1 |  | 154 | 1392 | 14156 | |  |
| 2.3 | Allocate | 3 |  | 193 | 1715 | 16908 | |  |
| 2.4 | De-allocate | 3 |  | 161 | 1379 | 13611 | |  |
| 2.5 | Allocate | 3 | x | 188 | 1754 | 16672 | |  |
| 2.6 | De-allocate | 3 | x | 154 | 1391 | 13286 | |  |

**3. Hyperty messaging**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No** | **Test** | **Duration in milli-seconds for different no. of iterations** | | | **Comment** | |
| **100** | **1000** | **10000** | |  |
| 3.1 | 100 Byte payload | 270 | 1557 | 14493 | |  |
| 3.2 | 1kByte payload | 185 | 1616 | 15853 | |  |
| 3.3 | 10kByte payload | 267 | 2227 | 22393 | |  |

**4. Subscription and Object update publication**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No** | **Test** | **Duration in milli-seconds for different no. of subscribers** | | | **Comment** | |
| **100** | **200** | **1000** | |  |
| 4.1 | 100 Byte payload | 282 | 569 | **FAIL** | | Limited Websocket resources in browser |
| 4.2 | 1kByte payload | 398 | 581 | **FAIL** | | See above |
| 4.3 | 10kByte payload | 498 | 791 | **FAIL** | | See above |