# Examining current academic and industry Enterprise service bus knowledge and what an up-to-date testing framework could look like

Joakim Olsson and Johan Liljegren

Blekinge Institute of Technology
laggmonkei@gmail.com, datanizze@gmail.com

**Abstract.** Nowadays integration and interoperability can make or break an enterprise's business success. In the huge space that is software engineering a lot of ESBs have emerged but with vast differences in implementation, patterns and architectures.
To create order in this disarray studies have been made, features evaluated and performance measured. This is a good thing but it does not clear up all the confusion and adds another layer of confusion regarding the studies and tests themselves.
The aim of this thesis is to make an attempt of rectifying some of the disorder by first evaluating the current body of knowledge and to provide a humble attempt of making a transparent test framework which could be used for a more coherent ESB evaluation.

## 1 Introduction

Since the beginning of software engineering the industry have come a long way in developing applications and platforms. This progress has been greatly beneficial for everyday tasks and for fulfilling industrial needs. The progress has been staggering but in some areas the industry have had a harder time. The issue lies in connecting these applications and platforms, making them communicate with each others. This is where the Enterprise Service Bus (ESB) [1] comes in. To address these integration issues terms like Enterprise Application Integration (EAI) [2] and Service Oriented Architecture (SOA) [3] have been visualized in the past. The ESB is a "next step" in this direction, taking what's good from SOA and EAI to bring a more complete platform for integration.

Early manual integration between different platforms (often referred to as "independent islands of computing") was done by point to point integration, that is, for each new platform to integrate all existing platforms had to build separate interfaces to communicate with the new one. This is a very cumbersome way to do it resulting in messy integration ending up with a lot of dependencies between all nodes. This type of integration is referred to as "spaghetti code" [4] since connections are made from everything to everything. This is where the ESB comes in. The ESB acts a mediator, translator and monitor amongst many other things. The main task for the ESB is to step in and take over communications

between all involved platforms. This is done by changing the involved platforms so they connect to the ESB and only to the ESB [5]. The main advantages of this is that the esb takes care of where to send data between the platforms making the platforms agnostic to what to send in regards to formatting/data structure and where, they just send it to the ESB and the ESB takes care of forwarding it to the right destination with the right data at the right place in the right language.

The research focus will be on how to write a test framework to make it easier for others to test different ESBs in a transparent and unbiased way. Another focus is to evaluate what is currently available in both academia and in industry to provide a review of what has been done and how is has been done.

When deciding what ESB to use a lot of parameters needs to be investigated, ease of use, documentation, features and performance are a couple of things needed to make an educated decision of what to use. This thesis focuses not on providing a list of which to use but to make it easier for others to make their own decision depending on their specific needs.

The nature of this thesis is to clearly define what is available regarding ESB testing and how it could be done in a more reproducable way. This reproducability is achieved by uploading all source code to a repository on github [6]. Github makes future testing easy and reliable for testers and readers since there is a clearly visible history covering the entire source code.

We will review literature available on the Internet in order to get a good understanding of the current body of knowledge and ascertain a need for increasing that body of knowledge. This also allows us to review how previous performance measurements and comparisons have been done in this field of software design which will ensure that our test framework does not deviate or alienate previous research as well as has a well anchored position in the current software design community.

The expected outcome of this thesis is to provide a defined test framework with sample results from a controlled test environment and to give a full review of the current body of knowledge regarding ESB solutions.

This thesis is aimed for individuals or enterprises that are in the position that they are starting to look at acquiring a integration platform and as such are interested in knowing which is the best performing ESB.

The thesis starts of with a literature review that examines the current body of knowledge in academia and industry, ending in a summary and conclusion regarding what tests this body contain. After the literature review we present our version of a testing framework followed by the results gathered from a live run of said framework. The whole thing is rounded up with a conclusion discussing the answers to the research questions asked, the contribution and proposed future work.

## 2 Background

Enterprise Service Buses (ESB) provides a platform for integration, connecting existing platforms and products with familiar protocols [1]. Before integration became an integral part of developing systems these now integrated platforms and products where operating on their own, without any interaction with other platforms/products. This can be viewed as "independent islands of computing". The ESBs task is to bring communication between these islands thus integrating them.

ESBs are pivotal in software integration and are becoming extremely important in company software solutions and infrastructures, future and ancient alike [7].

As such it is very important to accurately and repeatedly measure the performance of integration solutions available. There are currently several open source options available [8] and the current body of knowledge regarding their different performance aspects is severely lacking.

### 2.1 Core functionality

There are several different views on what the core functionalities of an ESB are. The most comprehensive list we have found was from Jieming [9].

**Location transparency**
: Seperates the service consumer from the service provider allowing the provider to centralize hardware while still providing regional services.

**Transport protocol conversion**
: Understanding several communication standards allows for several system to be integrated. Especially old legacy systems can be connected to new modern systems.

**Message transformation**
: Being able to translate between communication standards is essential for integrating different systems.

**Message routing**
: Being able to send an incoming message to the right system is vital for integrating systems as they can be seperated by location, age, protocol, country and many more criterias.

**Message enhancement**
: The ability to add information to incoming messages allow the ESB to connect services and send their combined message to other services, ultimately adding longevity to all services.

**Security**
: The ability to handle security is vital as the ESBs purpose is to connect several systems and as such is often the first and last defence.

**Monitoring and management**
: Efficiency is key and as such being able to monitor and manage an ESB is necessary

From this list we can extract a shorter more concise list of core performance functions.

**Direct proxy**
The ability to transfer incoming messages to a specific service.

**Routing**
Being able to look at incoming requests and based on their specific content forwarding them to different systems.

**Mediation/Transformation**
Translating between different inbound and outbound protocols.

These three functionalities are the very basic fundamental building blocks that all other ESB functionality depends on. An even better explanation of these core functions is that they represent an end to end flow trough the ESB. This means that one could simply replace the ESB with another ESB and the flow should be implementable without any disturbance to any other systems. Security features aswell as monitoring features are then layered ontop of this flow and as such they could be considered extras or ESB specific.

There has been some measurements and benchmarks done by the producers of the ESBs however these measurements have always been in favor of the ESB produced by the company making the measurements[10,11,12,13,14].

There has been some academic research done in the area [5] however the number of ESB's tested are quite limited when compared to the most up to date list of popular ESBs [8].

Most importantly they do not explicitly declare what versions they are using in their tests which makes their results hard to reproduce. Judging by the dates on which their paper was completed it is clear that all the ESBs that they use have received major upgrades which makes it even more important to produce new measurements.

The main purpose of this thesis is to increase the knowledge of differences between modern open source ESBs. This will hopefully lead to it being easier to further develop these software products and make it easier for end users to compare and evaluate based on their specific needs.

### 2.2 Current available ESBs

This section will list the currently available open source ESBs that we have found. The list is taken from a blog by Amar Mehta [8] and it the best collection of ESBs that we have found.

- JBoss [15]
- Apache ServiceMix [16]
- OpenESB [17]
- MuleESB [18]
- WSO2 [19]
- Fuse [20]
- Talend [21]
- Petals [22]
- Blackbird [23]
- Chainbuilder [24]
- Spagic [25]

Out of this list we have four ESBs that seem to be somewhat larger than the rest and those four are MuleESB, Fuse, JBoss and WSO2. We base this conclusion on the Forrester 2011 report [26] which seems to be a rather large survey, with very good industrial connections, covering many propriatory ESBs and the four open source ESBs mentioned above. We have also found three of the four in the tests performed by some of our other sources [10,27,28,11,14,5]. We can't say which of these four are the largest or the "best" overall solution as that is not a focus of this thesis nor very important when looking at performance. It would however be rather interesting to compare all four against each other in future works.

## 3 Research question and research design

This thesis is focused on performance testing and as such our research questions are aimed for that specific field.
The research questions are as follows:

- RQ1: What is the state of the art knowledge of ESBs in academic literature publications?
- RQ2: What is the state of the art knowledge of ESBs in industrialgi literature publications?
- RQ3: What are the components of a reproducable and accurate ESB performance testing framework?

RQ1-RQ2 will be answered through the literature review. RQ3 will be researched through literature review and answered by delivering our own testing framework.

### 3.1 Literature review design

The survey design consisted of searching the different search engines listed below with the listed search strings.

**Search engines used:**

- IEEE Xplore
- Google
- Google Scholar
- BTHs general search engine for academical documents

**Search strings used:**

- ESB
- Enterprise Service Bus
- Enterprise Application Integration
- Integration
- Integration measurement
- Integration comparison
- Mule esb
- WSO2
- Testing

During the literature survey abstracts were read and if the abstract suited our criteria (Evaluations of open source ESBs, performance measurements, ESB comparisons amongst others) a quick look through the actual content to see if the abstract did indeed sum up the paper in a correct and fair way. If then the quick view through the paper gave any value it was read more thoroughly and then added to the list of accepted papers to use.

## 4 Literature review results

The literature review has been focused on what the academic world knows and what the industrial world knows. Knowledge in this thesis is focused towards performance tests. First the academic sources will be presented and their tests summarized, followed by the same procedure focusing on the industry and finally the entire literature review will be concluded in an overarching summary.

### 4.1 Knowledge in academia

This section summarizes our findings in academia. It begins with a short presentation of papers found followed by a summary and analysis of the tests found within the papers.

"Enterprise Service Bus"[1] by Falko Menge is a paper that explains the fundamentals of an ESB as well as introduces Mule with an example. Published in 2007 its example has become obsolete and outdated however the fundamentals still hold true as seen by later papers such as "Research of Enterprise Application Integration Based-on ESB" [9] by Jieming Wu and Xiaoli Tao as well as "Integration of Distributed Enterprise Applications: A Survey" [29] by Wu He and Li Da Xu. These are papers that focus on describing the history and evolution of software integration. They were published in 2010 and give an in depth view on how an ESB operates.

"An Interoperability Study of ESB for C4I Systems" [30] by Abdullah Alghamdi, Muhammad Nasir, Iftikhar Ahmad and Khalid A. Nafjan and "Adopting and Evaluating Service Oriented Architecture in Industry" by Khalid Adam Nasr, Hans-Gerhard Gross and Arie van Deursen are papers showing the importance of ESBs in the modern world. Published in 2010 they represent a modern necessity for ESBs in industry.

"Service-Oriented Performance Modeling the MULE Enterprise Service Bus (ESB) Loan Broker Application " [31] by Paul Brebner is a paper from 2009 going trough how the author builds a integration solution and tests it, however the author tests the entire solution which isn't general enough for what we consider a performance test of an ESB. The author does however discuss some aspects of testing that are vital such as how and where to measure.

"Evaluating Open Source Enterprise Service Bus" [32] by F. J. Garcia-Jimenez and M. A. Martinez-Carreras, A. F. is the first paper found that performs a performance test however the test is limited in variation and magnitude. The paper was published in 2010 which makes the test results outdated since all ESBs in the test has received major updates since 2010.

"Enterprise Service Bus: A Performance Evaluation" [5] by Sanjay P. Ahuja and Amit Patel is the only paper found that performs a performance test directly aimed at the different ESBs with a varied test suite and varied measurements. Published in 2011 it is also the most recent paper found however all ESBs tested has received major updates since then and as such needs to be retested. Their test results seem to contain very low numbers when compared with the industrial tests containing numbers above 1000 TPS. This makes us question what system

and ESB configuration they have used but there is no way for us to check any errors on their part as they have not published these details.

The above papers has as per the literature review design first been identified by its abstract and then read in detail to find those that describe the concept of an ESB and those that test ESBs in various ways. We have selected papers according to three topics. First is a basic ESB understanding and explanation in order to assert an understanding of how an ESB works and its history. We consider this important since in order to understand the importance of a software it is imperative one has a firm understanding of its fundamentals. After that fundamental has been established we concentrated on papers that shows a use for ESBs in real world industry and organizations. This in order to convey a sense of the use and importance of ESBs and research concerning it. Finally we focused on papers that performs any kind of tests in order to establish an understanding of the current state of the art knowledge in the academic world. Most important were papers testing ESBs and not overall solutions, performance and not subjective feature checks.

**Academia summary** This section is a summary of tests and scenarios found in our academic sources.

| Reference to paper | Year | Performance | Evaluation | ESBs |
|---|---|---|---|---|
| Nasr [33] | 2010 | - | - | - |
| Alghamdi [30] | 2010 | - | X | Mule, Glassfish, Fuse |
| Sanjay [5] | 2011 | X | X | mule, servicemix, wso2 |
| Garcia [32] | 2010 | X | X | Mule, Fuse, Petals |
| He [29] | 2011 | - | - | - |
| Jieming [9] | 2010 | - | - | - |
| Brebner [31] | 2009 | - | - | Mule |

This table is a collection of the sources we have found and the essential information they contain.

| Test scenarios | Scenario 1 | Scenario 2 | Scenario 3 |
|---|---|---|---|
| Sanjay [5] | Direct proxy | Routing | Mediation |
| Garcia [32] | Direct proxy with security header | JMS bus | - |

This table is a collection of the scenarios performed in the sources we have found.

| Metrics | Client | ESB | Web service |
|---|---|---|---|
| CPU | - | Sanjay [5] | Sanjay [5] |
| Throughput | Sanjay [5] | - | - |
| Response time | Garcia [32] Sanjay [5] | - | - |
| Standard deviation | Garcia [32] | - | - |

Sanjay [5] tests used a varied amount of users and payload while Garcia [32] only tested with an increasing amount of users.

Table 3: Summary of metrics captured

The fact that we have only found two papers [5,32] performing tests is a very clear indication that the academic world have room for more performance tests/analysis on ESBs. The ones we have found are all considered old in a rapidly changing world and all but one [5] performs a test that we find adequately focusing on performance. It would also seem that Sanjay [5] have looked at the tests performed in the industry [10,27,28,14] as the tests and metrics captured closely align with the industrial tests however the numbers measured are wastly different. This shows the importance of publishing hardware specifications and tools configuration so following papers can reproduce and analyze data more accurately and improve on the testing framework. Sanjay have added a CPU measurement on the ESB in order to measure relative performance between ESBs allowing the same throughput per CPU load percentage to be calculated.

The tests performed by Garcia [32] contribute by adding security but the tests themselves doesn't test the core functions enough for us to use them in our own framework. Adding security on top of tests is a possible addition to the framework in the future but for now its at a higher technical level then we're aiming for in this thesis.

### 4.2 Knowledge in industry

This section will summarize our findings in the industry. Beginning with a short presentation of papers found followed by a summary and analysis of the tests found within the papers.

First of all we found an article by A. Mehta [8] which listed what the author thought were the best ESBs available in 2011. This is important as it provides us with a list of ESBs that we can pick and choose from when selecting candidates for our performance test and it provides us with focus points on where to start searching for performance tests performed by ESBs themselves. This is also the most recent article of its kin, as well as the largest that we found.

In 2007 WSO2 started with a series of three performance tests against other leading ESBs [10,27,28]. In the third they compare against Mule ESB which in turn responds with a performance test of their own from Mule [14]. The tests performed in these tests are sensible and aim to present an understanding of how the different ESBs perform while doing basic tasks. A problem however is that the results are from 2008 and later which mean that they are extremely out of date and maybe even more important is that they are very biased.

Mule has published a few articles that are not performance test but instead some kind of sales comparison with a number of other ESBs[11,12,13]. But we feel its important to include this anyway as we feel it shows a direction from doing performance tests to doing some sort of vague sales pitch without any numbers to back up their claims.

And finally we have read the Forrester report [26] from 2011 which is a industry initiated report that further visualizes the need for ESBs in modern software development, It is not performance oriented and includes non open-source ESBs.

**Industry summary** This section is a summary of tests and scenarios found in our industrial sources.

| Reference to paper | Year | Performance | Evaluation | ESBs |
|---|---|---|---|---|
| Falko [1] | 2007 | - | - | - |
| Fenner [7] | 2003 | - | - | - |
| WSO2 [10] | 2007 | X | - | WSO2 |
| WSO2 [27] | 2007 | X | - | WSO2, Mule, Servicemix |
| WSO2 [28] | 2007 | X | - | WSO2, Proprietary, Mule, WSO2, Servicemix |
| Mulesoft [14] | 2008 | X | - | Mule, WSO2 |
| Mulesoft [11] | 2010 | - | X | Mule, Jboss ESB |
| Mulesoft [12] | 2010 | - | X | Mule, Glassfish |
| Mulesoft [13] | 2010 | - | X | Mule, Servicemix |
| Forrester [26] | 2011 | - | X | Proprietary, Fuse, WSO2, Mule and more |

This table is a collection of the sources we have found and the essential information they contain.

| Test scenarios | Scenario 1 | Scenario 2 | Scenario 3 |
|---|---|---|---|
| Mulesoft [14] | Direct proxy | Routing | Mediation |
| WSO2 [10] | Direct proxy | Routing | Mediation |
| WSO2 [27] | Direct proxy | Routing | Mediation |
| WSO2 [28] | Direct proxy | Routing | Mediation |

This table is a collection of the scenarios performed in the sources we have found.

| Metrics | Client | ESB | Web service |
|---|---|---|---|
| Throughput | Mulesoft [14] WSO2 [27,28] | - | - |
| Response time | WSO2 [10] | - | - |

All tests were performed with varied amounts of user and payload.

Table 6: Summary of metrics captured

The industry seem to have a good understanding of how to test their products however they do not seem to update their results and as such they hold little value. There also seems to be quite alot of rivalry in the different tests. It seems WSO2 started by doing a series of performance tests [10,27,28] where Mule did not perform as well as Mule themselves think they should and such they did their own test [14] where the results pointed in their favour. This shows a need for a independent third party to perform the testing as it will otherwise turn in to a slugfest where no results can be trusted.

### 4.3 Literature review results

In this section we will answer RQ 1 and 2 aswell as provide a foundation for answering RQ3.

It is apparent that the academic world is lacking performance tests regarding the ESB field of software engineering. We found one paper by Sanjay [5] that produced a test suite with a good enough scope and focus. It is however becoming outdated due to the fast pace of the industry and there are no collaborating papers supporting the results produced and as such it by itself hold little value. We have also during our own testing found Sanjays results quite odd and since they have not presented the environment in which they performed the tests it's impossible to figure out why their numbers are strange. The industry has made a good effort in providing sensible tests that focuses on the basic performance but those results are old, outdated and most importantly biased as the different companies are direct competitors.

We believe that [5] and [10,27,28,14] still holds some merit in regards to how tests should be performed but the actual results they have produced have not been confirmed by other independent sources and are now to old for reproduction. It is therefore our conclusion in regards to RQ1-2 that the current state of the art knowledge is inaccurate and outdated however there is enough history for us to be able to start fresh. We will focus on the scenarios found in [10,27,28,14,5] as those seem to be focused on the core functionality and that is the best place to start. These scenarios should also allow for atleast a comparison against older data and some conclusions regarding improvements in performance might be possible because of that. We will in our test framework provide an up to date framework for performing these tests and most importantly we will provide the source code needed to replicate and analyze the tests.

# 5 Testing framework

This testing framework aims to answer RQ3. The goal is to produce a transparent and sound way of testing core functionality (see section 2.1) performance of any ESB. We have gathered much inspiration from Sanjay [5] regarding what measurements and tests should be performed however an even larger concern of ours is to provide source code and transparency in order for the tests to be repeatable as well as having the ability to discuss how to improve testing which Sanjay misses completely.

## 5.1 Measurements

This section will describe what metrics will be collected and why.

| | Metrics captured |
|---|---|
| Client | Response time and throughput |
| ESB | CPU |
| Web service | CPU |

Table 7: Metric captured

It is vital to monitor the overall status of all systems involved in the test in order to detect possible chokepoints like network becoming maxed out thus severly limiting the test results.

CPU usage for the web service is important to measure because the web service should not be bottlenecking anything, it is just there to simulate a real back end application and thus has very little to do with ESB testing except for being there as a helper for the whole integrated system. CPU usage for the ESB is important if two or more ESB solutions are tested against each other for when calculating relative performance where CPU usage relative to how much is processed plays a great part.

**Response time** The amount of time elapsed since a request was sent to the time a response was received. The response time is measured because this is what a single user will be affected by the most, if the response time is very low the user will perceive the system as very fast and vice versa if the response time is too high.

**Throughput** The amount of successful transactions performed per second (tps). A transaction is counted as successful if the response matches expected values. The value of throughput lies in how many users the system can serve in a specific timeframe. If the throughput is high the system will be able to handle many simultaneous users making the system efficient at handling a high load.

Response time and throughput are the most interesting numbers to investigate as they are very important for end users and systems. The more users a system can handle and the faster it does so the higher value the system will get in terms of scalability and performance per processing unit which in turn equals a lower operating cost where the system is able to handle a high amount of users whilst keeping the costs down.

CPU will be measured in order to be able to make sure no hardware limitations occur during testing. The CPU on the web service is only collected to make sure that it doesn't act as a bottleneck while testing. Although this might not be the case in real world scenarios the aim is not to cap any hardware limitations because that would not give fair test results since the hardware would limit the performance of the system being tested and since the framework is supposed to give the involved system(s) free roam hardware limitations would be in straight contrast to what the framework is about.

## 6 Test walktrough

The tests described below are aimed at measuring the very basic roles of any ESB. The tests are very simple with the specific goal of producing a baseline for future tests. These basic tests have been chosen since anyone with the development knowledge of an ESB should be able to produce an ESB project capable of delivering a runnable project which exposes these basic funtionalities. Each of these tests have been run in two different ways. The first one is by letting a single user (client) send a payload which is increased each run where they payload size varies, 1KB, 50KB, 100KB, 500KB and 1MB. The other way is having a fixed payload size of 100KB and a varying amount of concurrent users (clients) in the steps 1, 20, 40, 80, 160 and 320 users.

### 6.1 Pure throughput

No manipulation of the data will be done by the ESB in this test. The ESB will simply forward all request to the target web service (as fast as possible) (fig.1). This test could simulate a service where the ESB exposes an inbound endpoint for other an system to connect to, modularizing which service is running in the backend thus giving a separation of concern where the requesting client does not have to know about the responding web service, just that a web service is made available by the ESB. A comparative test can be performed here as well and that is to not use the ESB which will show what performance impact the ESB adds.

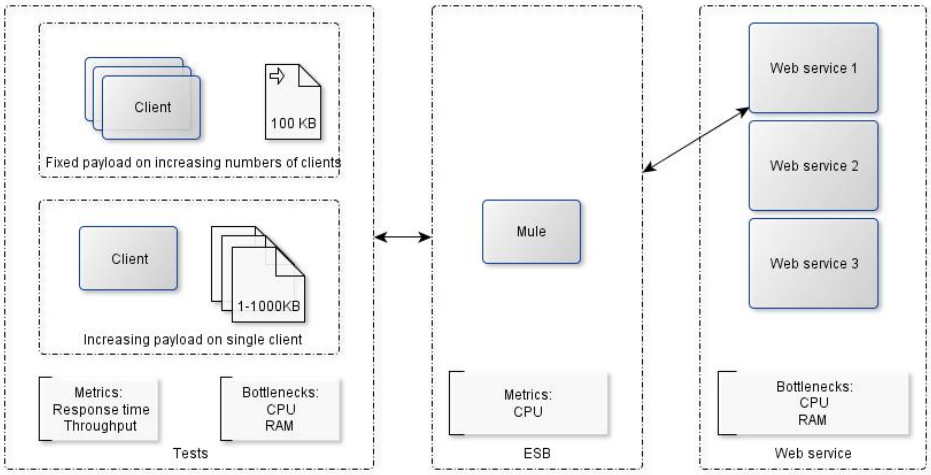



Fig. 1: Direct proxy structural diagram

### 6.2 Routing

In this test the ESB will check the incoming request from the Client and dependning on the context, send the request to an appropriate web service which will append some data and return the request to the ESB which will send the response back to the Client (fig.2). This represents the ability to have several systems behind an ESB all showing the same front to the clients. This test is deemed basic because it is what a ESB should do, separate the integrated systems from each other, stepping in as a "middle-man" directing requests on behalf of the systems being integrated making the need for changing the systems minimal and just focus on the ESB. A test like this could simulate a load balancer where the ESB is a front for two or more systems each running an instance of the same service. Another scenario this could simulate is different systems handling different (but similar) data, for example one system behind the ESB is handling user registrations and another one is handling user logins. The ESB can then look at the request payload and decide what user interaction is happening and thus route the request to the appropriate system.

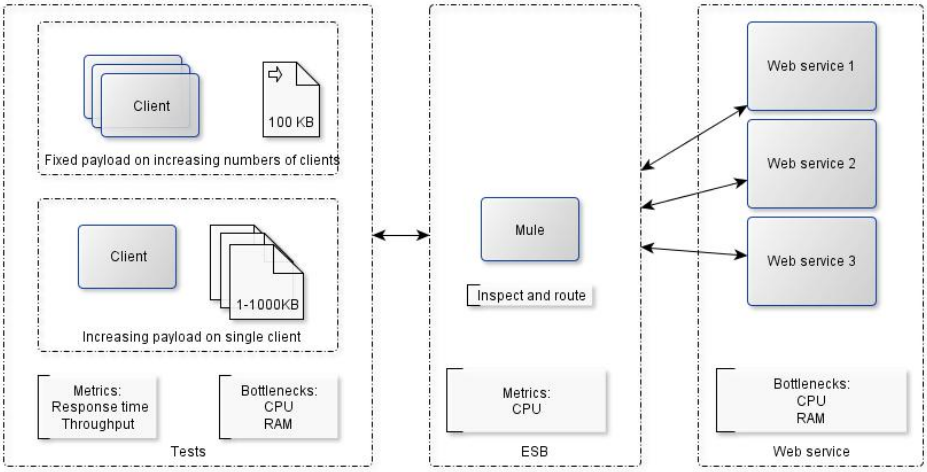



Fig. 2: Routing structural diagram

### 6.3 Message transformation

The ESB will convert an incoming request to a different protocoll and send it to a web service which will append some data and return the request to the ESB which will transform it back into the format the Client originally sent it (fig.3). Transformation is a big deal for any ESB since two systems to be integrated probably won't speak the same language (XML, json etc) and even if they do they may not have the same data format or structure. Of course the systems need to be able to communicate with each others, without an ESB this would become cumbersome since all involved systems would have to be changed in order to understand what the others where saying, just imagine an old system written in COBOL or Assembler and making it communicate with a new system like a web service using REST [34]. Integration like this cost both money and time and probably won't be future friendly with regards to maintenance. This could make an ESB invaluable since the ESB would take care of receiving the incoming request in one protocoll, decide where the request is to be sent, transform it to a protocoll which the receiving system can understand, get a response back and transforming it to a protocoll the original system which sent the request will understand and this without even touching the code for the two systems talking to each other.

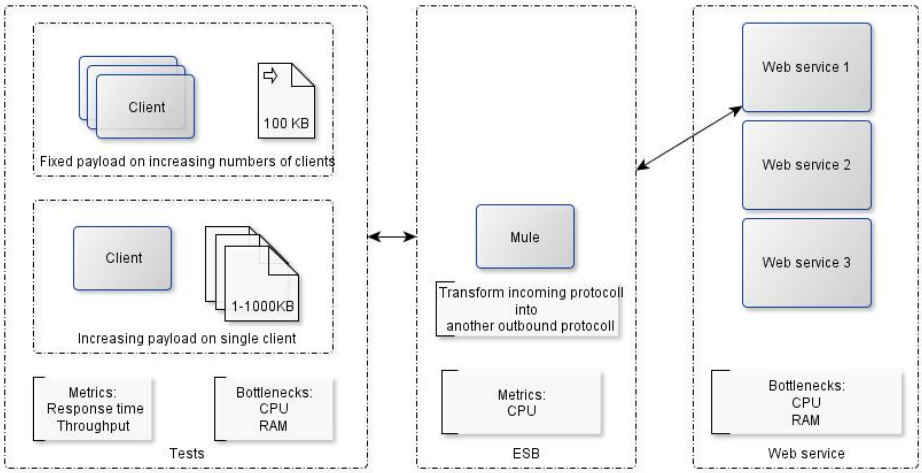



Fig. 3: Transformation structural diagram

### 6.4 Artifacts and tools

This section describes the artifacts and tools used to produce test loads and capture metrics aswell as other hardware environments.

- Client: Grinder (v3.8) [35]
- ESB: Mule (v3.2.1) [36]
- Web service: Jax-WS (v2.2) [37]
- OS: Windows 7, 6.1.7600 build 7600

Table 8: Software and tools

| Component | Specification |
|---|---|
| CPU | Intel Core i7 920 @ 2.67GHz |
| RAM | 6,00 GB Triple-Channel DDR3 @ 533MHz (7-7-7-20) |
| Network | Intel(R) 82567LM-2 Gigabit Network Connection |
| Motherboard | Intel Corporation DX58SO (J1PR) |
| Graphics | 256MB GeForce 7600 GS (MSI) |
| Hard Drive | 488GB Western Digital WDC WD5001AALS-00L3B2 ATA Device (SATA) |

Table 9: Hardware configuration

In order to minimize the amount of factors that interfere in the tests we recommend having at least three similar state of the art computers, connected to a high-speed network, essential. It might not be of the greatest importance that the computers are state of the art but in order to not reach a hardware ceiling while testing, such machines are recommended. What's most important is that one computer is designated to run ESBs, one is designated to generate traffic(called Client) while the others are simple servers (called Web service) are running a simple application which echoes everything sent to it with an interface using jax-ws to expose the service to soap/XML clients which is the interface used by grinder/the esb to send requests to the web service.

This separating and designating of roles to machines minimizes different hardware affecting test results as the same machines perform the same roles in all tests and the only thing changed is the ESB.

It also means that if other machines are used in other tests the data produced can be compared to ours and as such validate the data or identify faults in the tests. This validation can be done in two ways. First is to run the same software versions and compare the results. They should be similar deviating only in magnitude. The second way is if using a newer or older software version the values, when put in a graph, will either have the same shape or show areas

in which performance has changed, if it is the same shape but the magnitude is higher then that is most likely caused by faster machines being used and vice versa if its the same magnitude except in certain areas then that shows an improvement in the software.

The framework is available for others by simply following the base tests described in section 5.1. Those test can be considered as a basic starting point and expanding these tests with more complicated ones is more than welcome and is encouraged. The configuration files and all source code used in the framework can be found on github at https://github.com/Datanizze/korsdrag-thesis-files

### 6.5 Variables and variable control

No limitiations has been forseen except for hardware and network. Hardware and network loads should be monitored so they are not close too 100% as that would mean there is a major bottleneck present. All involved computers were running the same operating system.

### 6.6 Validity threats

This section discusses different validity threats.

**Inefficient code** We are not expert in the various programming languages and systems used to perform these tests and as such our code might be inefficient and even wrong. Inefficient code is not a problem as long as the same code is used in all tests. Erroneous code however is unmaintainable code and as such will require to be changed in the future and that will most likely affect the test results and test history. The possibilities of erroneous code has been limited by focusing the tests on very simple and basic functionality that doesn't require expert knowledge of the ESB in order to get results. By actually providing the source code we have opened upthe possibility for improvements and additions of more advanced tests which we feel is extremely important for a consistent testing framework used in academia and industry.

**Misconfigured tools** Tools such as grinder and our webservice could most certainly be configured better. Especially grinder should be looked at as there are more subtle parameters that can be set that could impact testing results such as sleepTime that pauses the client slightly before sending a new request.

**Different operating systems** Different operating could alter the results of the framework slightly. We used a Windows 7 OS that is not optimized for network traffic like a tweaked *nix system could be. This shouldn't be that much of a problem as long as the tests are performed using the same machine with the same optimized OS. It could make it harder to compare results to previous research but it would not comprise the framework as long as the OS configuration is published with the testing results.

**Not using different measuring tools** We focused on using Grinder as a load generating client and measuring client but perhaps we should have tested different measuring clients in order to ascertain that Grinder produces correct results. We did look at other tools before deciding on Grinder however and Grinder seemed to be the easier and more powerful tool to use.

### 6.7 Test results and analysis

This section describes our attempt at running the testing framework with an analysis of the results. First up is the hardware specification on the five computers used, the scenarios where run with one computer acting as the client , one as the ESB which in this case was mule and three as web services (see table 9). The computers were connected to an isolated network environment with a 1Gbps speed limit.

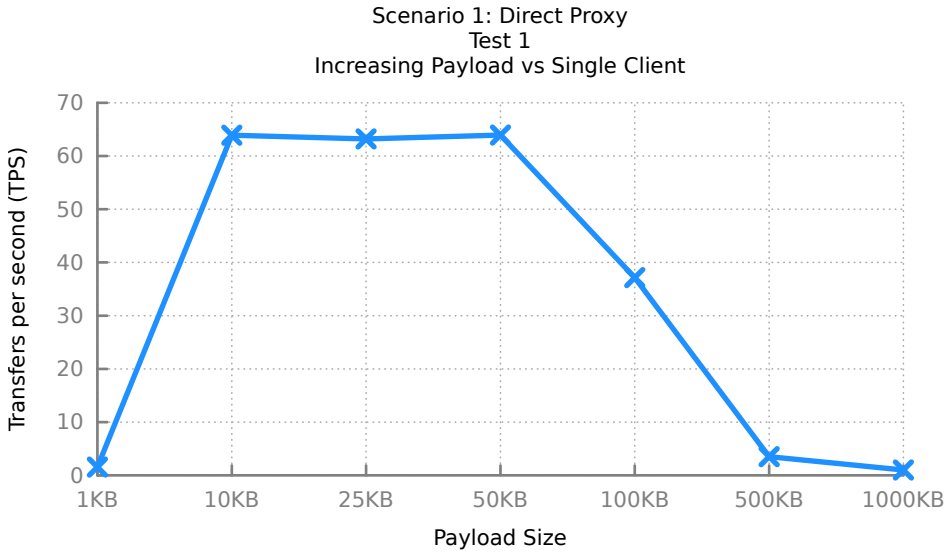
As described in table 8, Grinder was used for simulating clients (users), the ESB in this case was Mule and the web service running on three different computers used JAX-WS.

The results have been obtained by running each test in a period of 5 minutes, that is, 5 minutes for shift in test parameters (changing the number of clients and/or the payload size). This showed to be a sufficient amount of time since the amount of requests made in each test ranged from at least 350 to more than 30 000 which was deemed sufficient for liable test results. All data collected in the following sections were done by grinder.

In the following sections the results have been analyzed for each scenario summing it up with an overall analysis of the performed test scenarios.

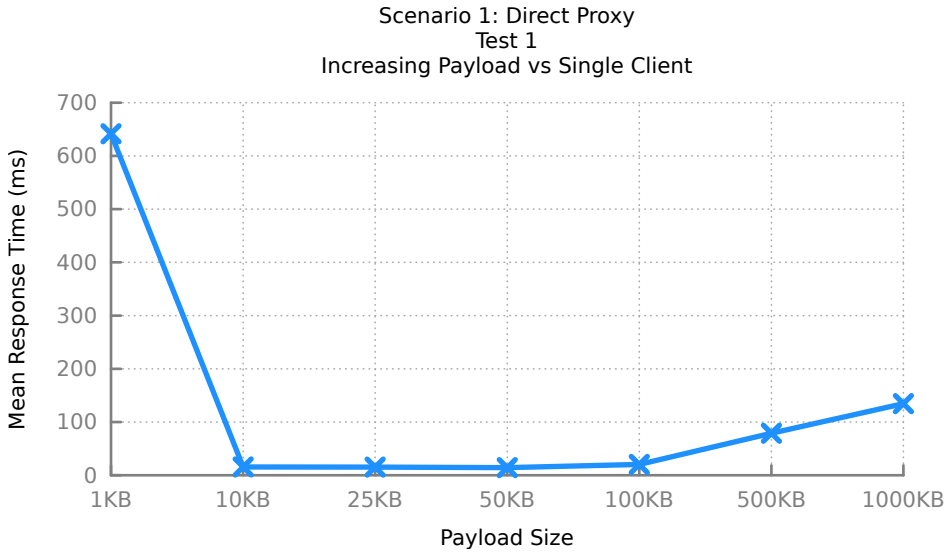
**Direct proxy results/analysis** The first test for the proxy with a single client sending an increased amount of datai (Fig. 4). Mule seems to handle this quite well, peaking at about 64 transfers per second (TPS) which could be to be a limit in mule's cxf proxy endpoint which was used. When a single client is sending a very small payload (1KB) it seems like our mule project implementation does not handle this very well since all scenarios where affected by this, limiting the throughput to about two TPS. Mule is handling the direct proxy well and it's handling regular sized payloads with ease only beginning to throttle when the payload size is growing beyond 50KB (Fig. 6).

Fig. 4: TPS for direct proxy.



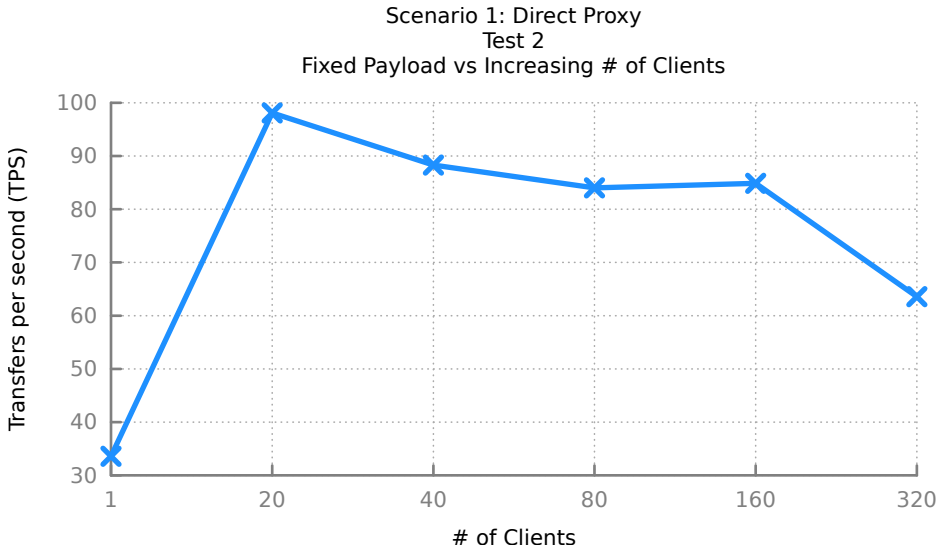

As seen in this figure mule is having problems when the single user is sending a small amount (1KB) of data. As the payload increases a bit (10-50KB) a peak at 64 TPS is reached and is then decreased when the payload reaches 100KB. When the payload size reaches 500KB mule is having som difficulties keeping up only giving a TPS of 5.

Fig. 5: Mean response time for direct proxy.



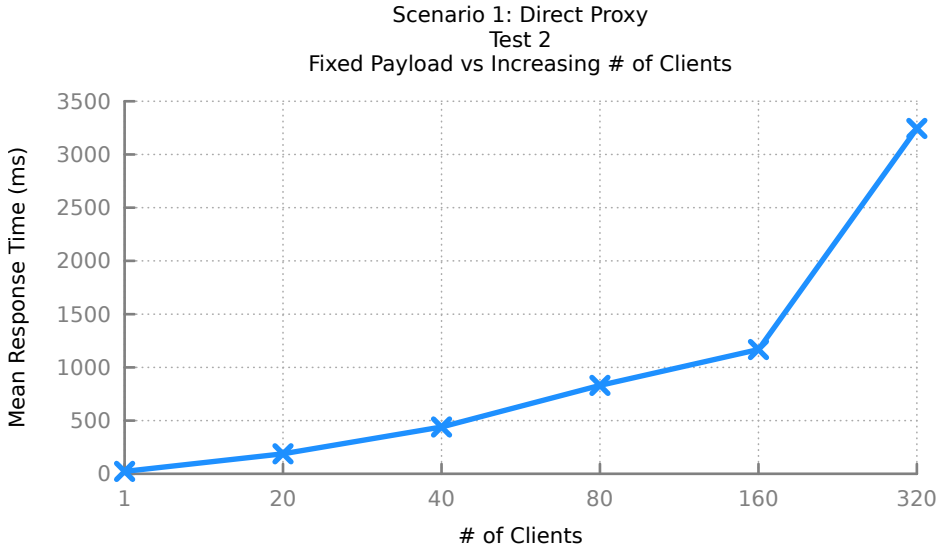

The mean response time for the proxy keeps low throughtout the entire test scenario where the largest peak is at the 1KB payload which has been attributed to a limitation or bad configuration in mule. It is only when the payload size reaches 500KB and beyond an increase in mean response time occurs.

Fig. 6: TPS for direct proxy.



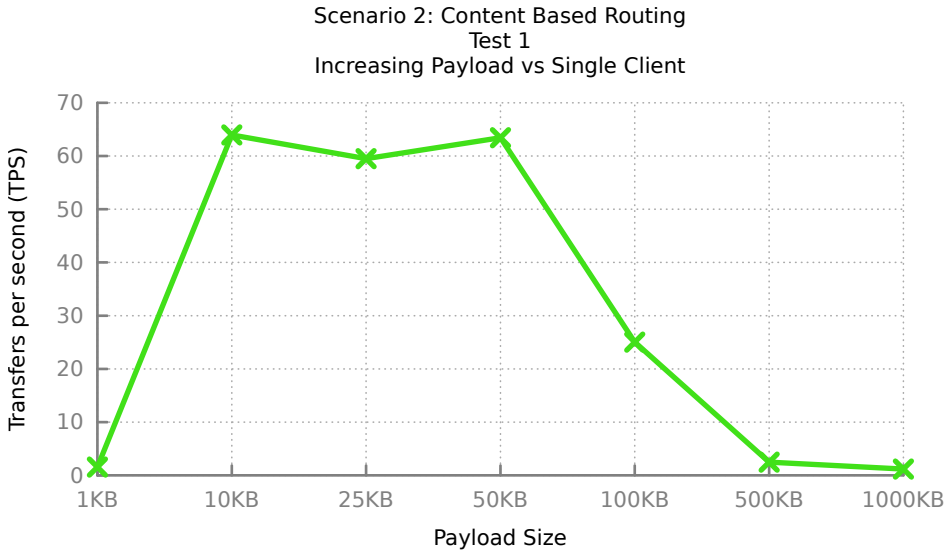

Using a fixed payload of 100KB and an increasing number of clients the throughput was peaking at 98 TPS and then decreasing at 320 concurrent users.

Fig. 7: Mean response time for direct proxy.



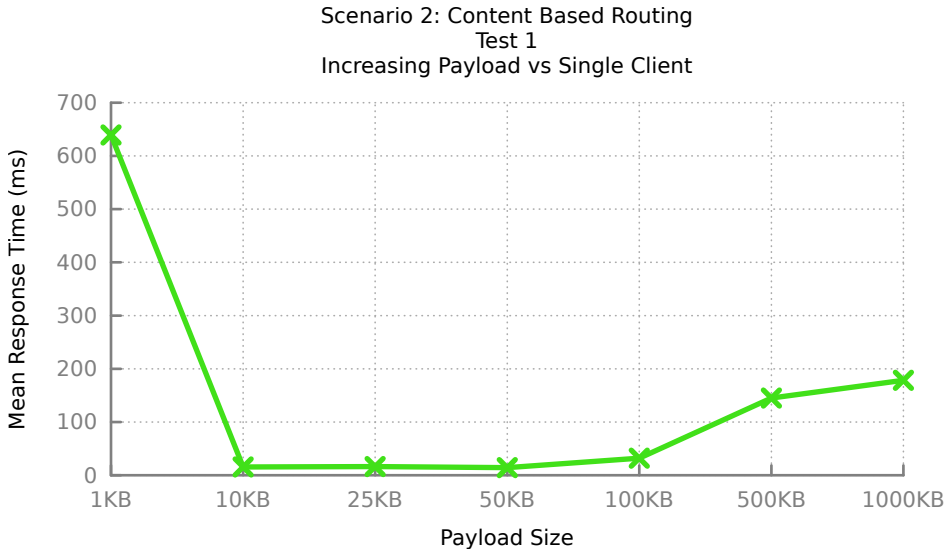

Mule handles the increasing number of clients well, keeping up with a reasonable response time only to slow down when the client count increases beyond 160.

Fig. 8: TPS for content-based routing.



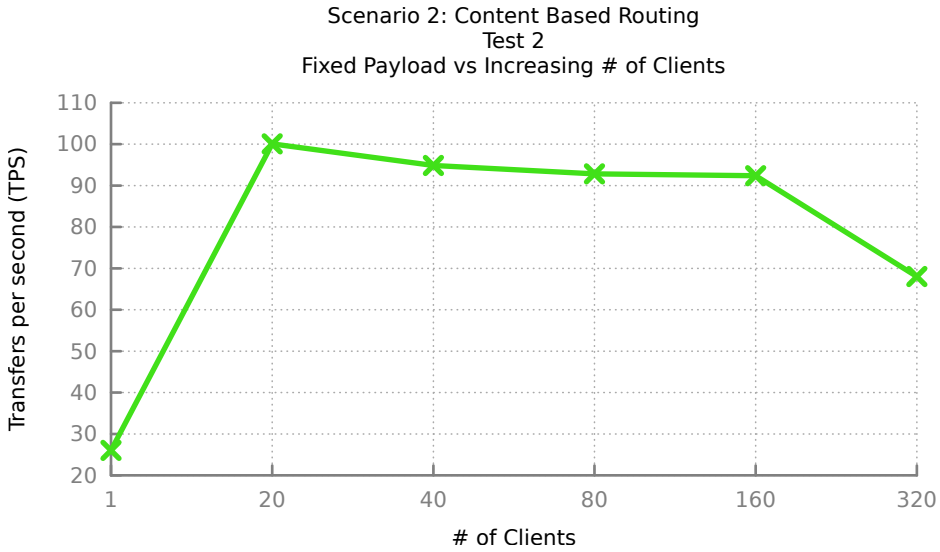

As seen here in the content based routing scenario the behavior of mule is similar to the direct proxy (Fig. 4), starting off slow again at 1KB to increase again at 10-50KB peaking at around 65 TPS and then going down again to about a third in TPS finishing off with 5 TPS at 500KB and 2 TPS at 1MB.

Fig. 9: Mean response time for content-based routing.



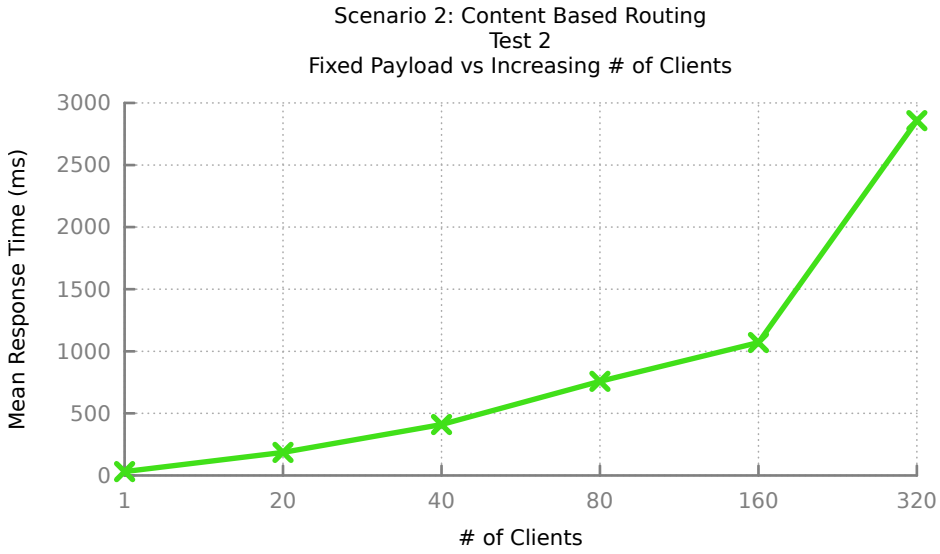

Similar to the mean response time seen in the direct proxy scenario (Fig. **??**) with the exception of being slightly higher at 500KB and 1MB caused by the fact that mule has to look at each request and make a decision of which web service to send the request to.

Fig. 10: TPS for content-based routing.



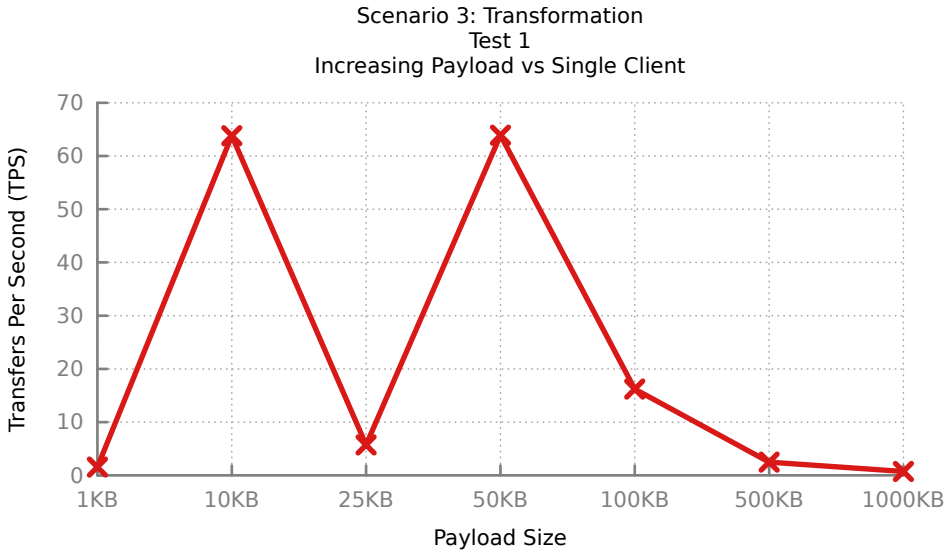

As with the direct proxy (Fig. 6) a single user sending 100KB is limited by the fact that a single user can only send so much requests in a short time frame. The content based routing scenario almost mirrors the direct proxy in this test as would be expected since mule is in addition to proxying just taking a look at a small payload to decide where to send it thus no major processing is done by mule.

Fig. 11: Mean response time for content-based routing.



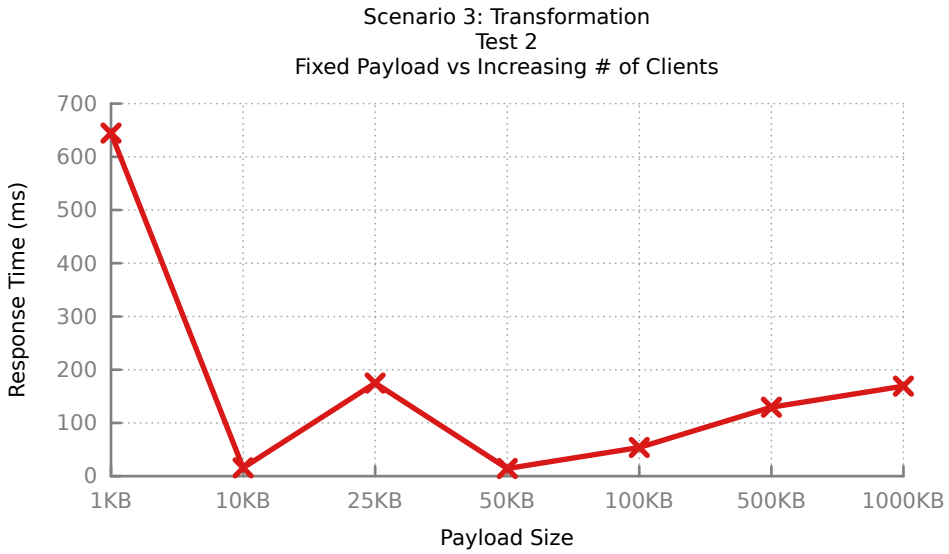

The mean response time for this test scenario almost mirrors the same test scenario for the direct proxy (Fig 7). An almost linear increas in response time can be seen as the number of concurrent users increases only to deviate when the user count peaks at 320 users.

Fig. 12: TPS for content transformation proxy.



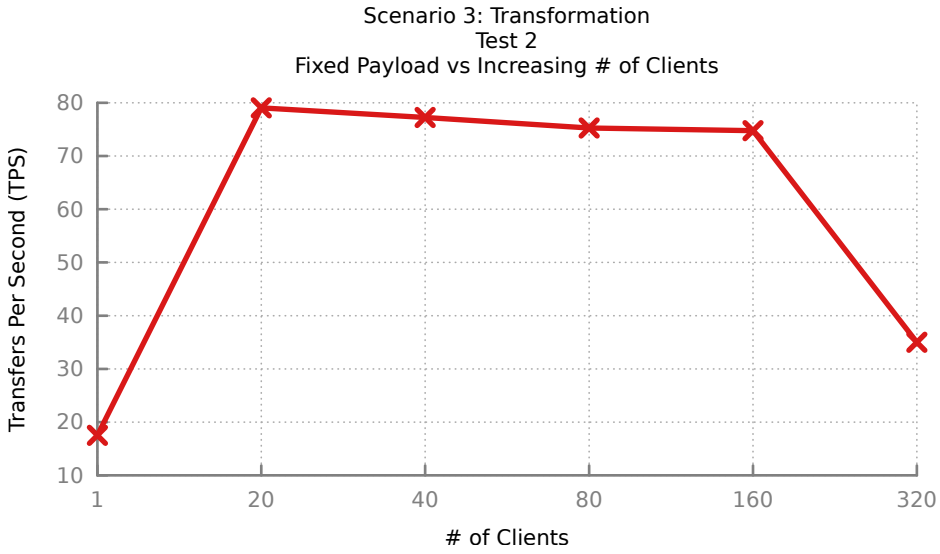

The first test with a single client sending an increasing amount of data in the transformation scenario almost mirrors the direct proxy except for when the client was sending a 25KB data payload. The 25KB part of this test was rerun three times with the same result. A conclusion was reached that this might be a bug in the mule software since the throughtput went up again when the payload was increased to 50KB. The only other thing noted was a higher decrease in throughput when the payload reached the larger sizes, this was concluded to be caused by mule having a hard time transforming such large amounts of data from json to xml and back again.

Fig. 13: Mean response time for content transformation proxy.




The graph here follows the earlier test scenario graphs (Fig. 5 and 9) with the exception of the odd 25KB behavior.

Fig. 14: TPS for content transformation proxy.




The second test in the transformation scenario shows that sending mule is quite capable of handling transformation of many requests. Note that the throughput is higher for 160 clients with this scenario than in the direct proxy scenario.

Fig. 15: Mean response time for content transformation proxy.


Mule is handling the transformation between JSON and XML well, mirroring the behavior of earlier response time results (Fig. 7 and 11) when not accounting for magnitude.

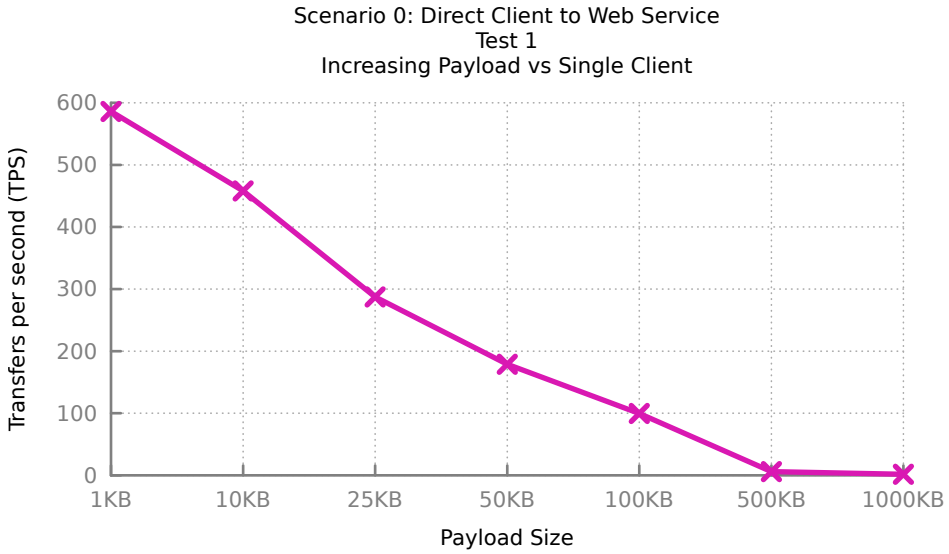
**Overall analysis of performed tests** The low throughput seen in all of the tests when a single user is sending a very small amount of data (1KB) (Fig. 4, **??** and 12) is something we found very strange. To get som clarity we ran all tests without involving mule, a direct connection between the client (Grinder) and the web service (EchoService). As seen in Figures 16, 17, 18, 19 the test results are quite different where the only similar graphs (ignoring magnitude) being the Mean response time.

No absolute conclusion about what caused the low performance when using mule was reached but we speculated about possible user errors or faulty configurations caused by deficient know-how about coding well-performing mule code.

We decided to run the direct proxy scenario without an ESB in order to see wether the low performance is because of hardware limitations or because of our ESB implementation.

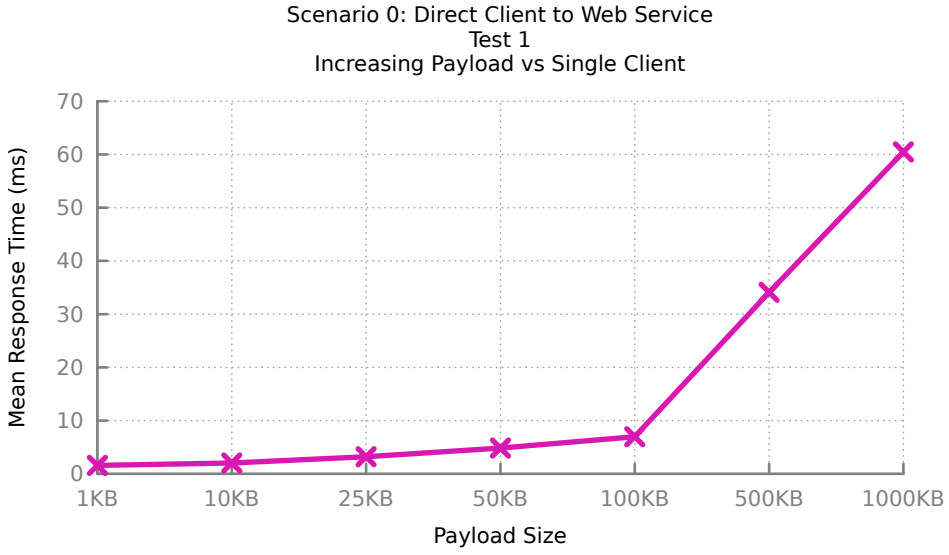
Below are the results from that test followed by an analysis.

Fig. 16: TPS for client direct to web service.




This graph is what one would expect in throughput where a small payload gives a high throughput and a larger payload decreases the throughput.

Fig. 17: Mean response time for client direct to web service.

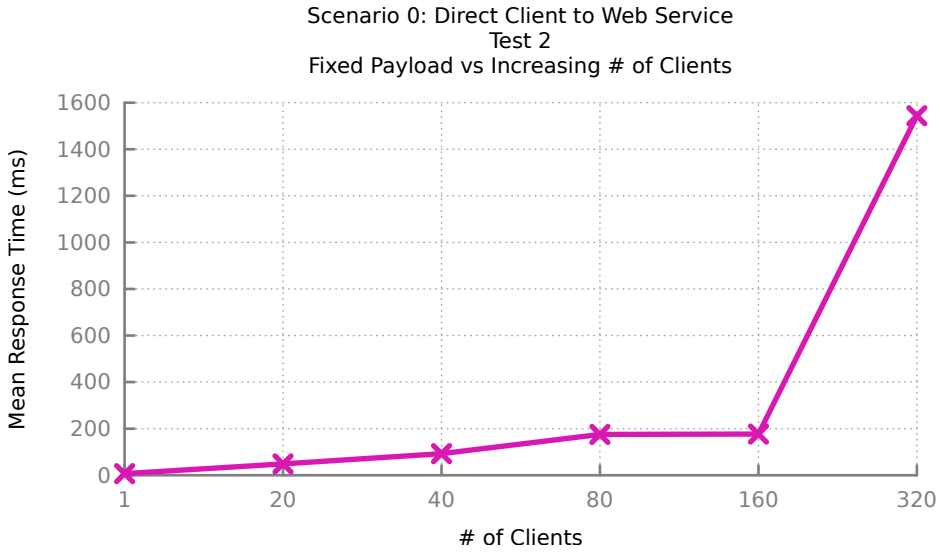



A very good mean repsonse time can be seen here when the payload size is 100KB and below. The response time increased many tenfold after that but is still well below 100ms.

Fig. 18: TPS for client direct to web service.

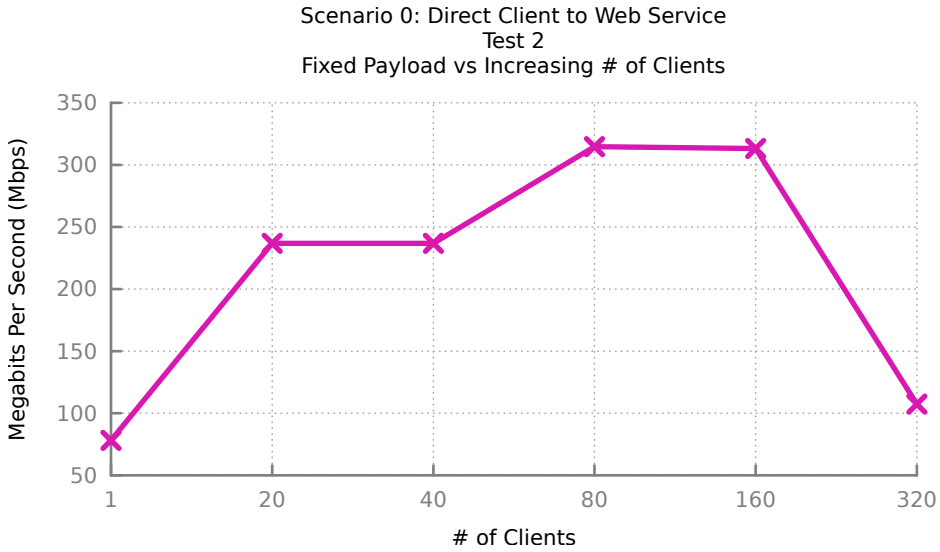

When the test scenario with a fixed payload of 100KB and an increasing amount of users were run we note that a much higher throughput is reached without Mule. Running the direct tests with 160+ users sending 100KB requests was throttled by the network, therefore the peak is at approximately 400 TPS.

Fig. 19: Mean response time for client direct to web service.




A mean response time below 200ms was kept up until 160 concurrent users. When the user count reached 320 the network got overloaded and thus could not keep up with all requests being sent at a proficient rate making the response time increase by a factor of eight.

Fig. 20: Average network usage for client direct to web service.




As seen in this graph the network is peaking at around 320 Megabits per second on average (which is only half the network capacity since the web service is echoing everything sent to it instantly doubling the network load).

Since the 1 user 1KB test's TPS as shown in figure 16 is much higher than in the direct proxy (fig. 4) our suspicion regarding a limitation/bug in mule seems to fit.

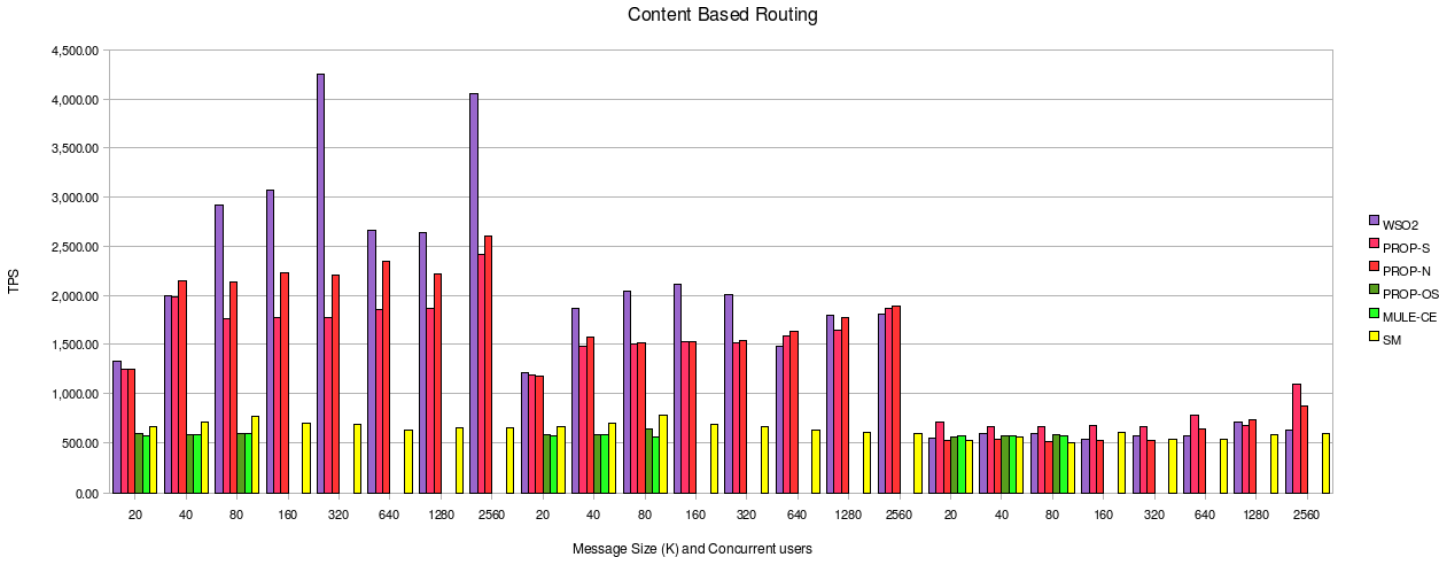
Noting these major differences in test results with and without mule we concluded that it is not necessarily our framework that needs refinement but that our skills programming an integration bus is lacking. This is why our framework matters, everyone won't be able to in a small period of time learn to be masters in each and every ESB being tested. Instead we want to delegate the integration bus part to others with the right competence to integrate with our endpoints (client and web service) in a way that makes optimal use of the ESB. This fits well with the valdity threat about inefficient code in relation to our tests since we are no experts at developing in mule.

# 7 Conclusion

The current body of knowledge is severely lacking in both academia and industry (see section: 4.3) . We found only two papers [5,32] doing performance tests in academia and a larger number of outdated and biased industrial papers [10,27,28,14]. There is enough papers for us to at least get a foundation of testing to build upon which is what we do in our test framework. The framework consists of a series of scenarios aimed at testing the core functionality of any ESB and we have used Mule ESB as verification that the framework works as expected.

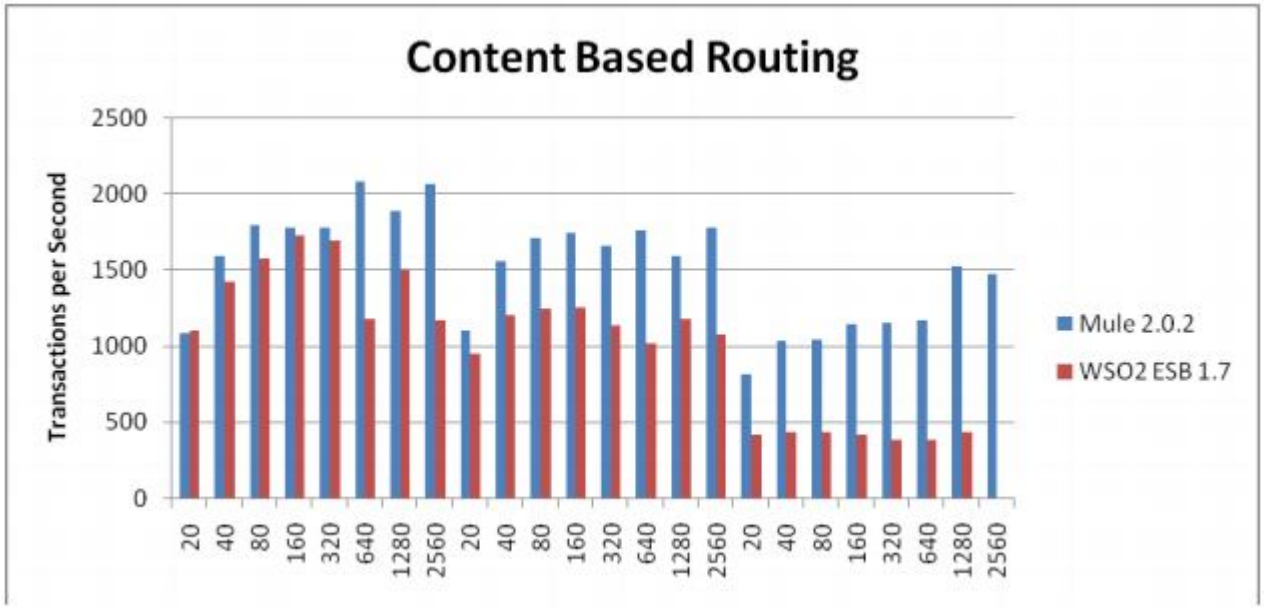
The largest conclusion done in this thesis is that the current body of knowledge seems to be severly lacking in both industry and academia. Most importantly the paper we have found [5] lack a reproducibility which is, possibly, even more important than an unbiased test. We have also started a humble attempt at creating a testing framework that could serve as a foundation for future tests. Below is an analysis of some of the tests performed by the industry and academic sources found.

Fig. 21: Image taken from WSO2s test [28] showing their content based routing measurements.




Payload is increasing (0.5KB, 1KB and 5KB) and the x-axis is numbers of clients sending data. All values seem to be above 500TPS with some tests not being able to complete on all ESB solutions.

Fig. 22: Image taken from Mules test [14] showing their content based routing measurements.

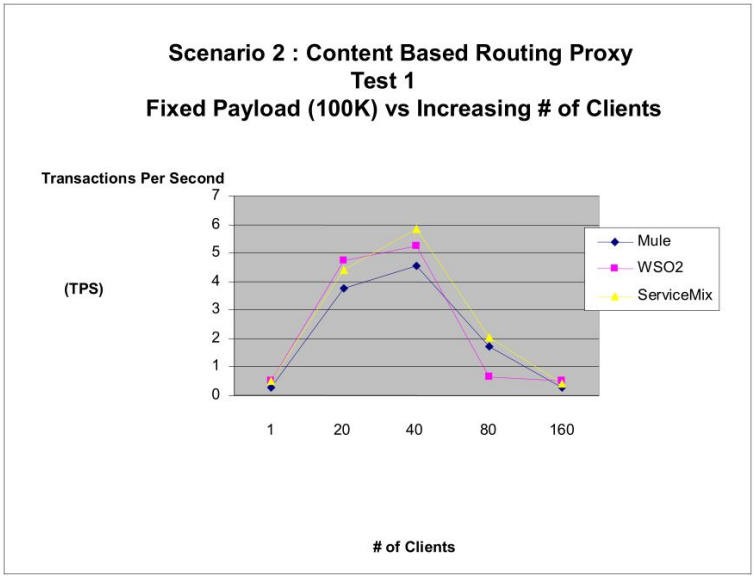



Payload is increasing (0.5KB, 1KB and 5KB) and the x-axis is numbers of clients sending data.

While looking at figure 21 we can clearly see that Mule has failed the majority of tests performed by WSO2 [28]. It is therefore not suprising that in the following figure from Mules test [14] shows WSO2 failing tests and performing far below Mule. This blame-game is why it's so important to have a unified performance testing framework that everyone can use as a performance index that can be trusted for its accuracy and unbiased measuring.

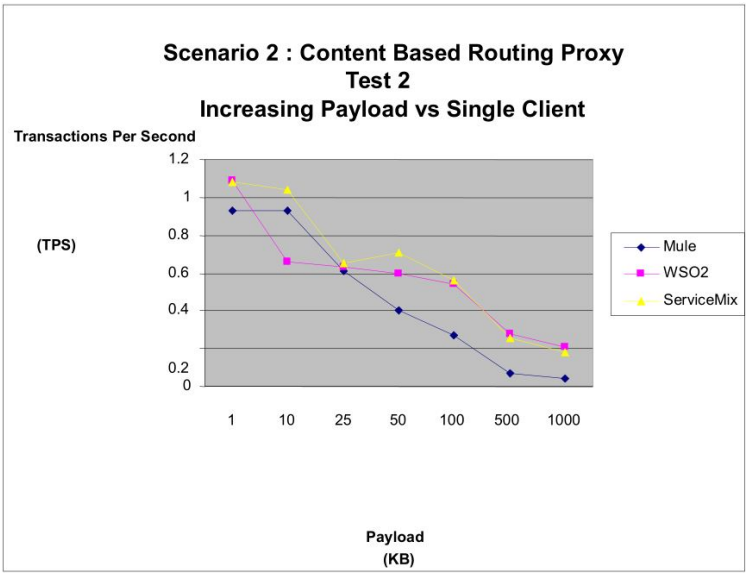
What is most suprising to us is that in these tests the TPS is above 500TPS and often above 1000TPS overall. If we look at the second "20" mark on the x-axis which is 1KB sent by 20 concurrent users both WSO2 and Mules tests show values between 500TPS and 1000TPS while in our own tests with an ESB we didn't even break 10TPS. Even without an ESB we could not achieve such high numbers which could mean that the configuration of grinder or the webservice needs to be finetuned. WSO2 and Mule also ran very short tests (Mule: 15s, WSO2: 1s) which would result in a rather low amount of data collected.

Fig. 23: Image taken from Sanjay [5] showing their fixed payload content based routing measurements.




Using a fixed payload of 100KB and an increasing number of clients the throughput was peaking at 4.5-6 TPS at 40 concurrent clients.

Fig. 24: Image taken from Sanjay [5] showing their increasing clients content based routing measurements.




Single client sending an increasingly larger payload with a TPS between 1.1 and 0.2 which are simply appallingly low numbers.

When looking at the results from Sanjay [5] in figure 23, 24 and the results from WSO2 and Mule [28,14] in figure 21 and 22 one can't help but wonder how Sanjay performed their tests. Even without the knowledge of the previous tests by WSO2 and Mule their numbers are so low that it seems impossible to consider them as correct yet there is no discussion regarding the validity of the data or their testing procedure. It doesn't help that they have not published their hardware setup nor their configuration and ESB source code which means that we cant even begin to hypothezise regarding their data.

Our results as seen in chapter 6.7 indicate to us that the framework needs more finetuning but that is a good start for future work. It is also a good indication that without professional help it is very hard to setup an ESB that performs aswell as the ESB manufacturers claim, we reach this conclusion since we have 6months of ESB deployment experience in a student project and still our implementation is far from the results in figure 22. The largest contribution done in this thesis is that this could be a starting point for making a testing framework that gives reproducable results and a unified way of testing ESB performance.

One of the more interesting suggestions we would like to make is to include the ESB manufacturers themselves and have them produce the ESB code that is then tested using the framework. This would allow a larger number of ESBs to be tested and the code would contain less faults due to the testers inexperience of a particular ESB. It would require some secrecy as otherwise the ESB manufacturers could optimize the code in an unnatural manner and that would skew the results.

Recieving optimized code from the manufacturers would also create a baseline that could be used inorder for the tests to be made more advanced and complex. Increased complexity of the tests could result in more accurate performance tests as the hardware is pushed closer to being maxed out. Finetuning of the tools used such as grinder could also be required before the framework is used on a larger scale.

The configuration files and all source code used in the framework can be found on github at https://github.com/Datanizze/korsdrag-thesis-files

## References


1. F. Menge, "Enterprise Service Bus." http://kanagwa.com/assets/21/Esb1.pdf, 2007.
2. L. Z. Yan Du, Wuliang Peng, "Enterprise Application Integration: An Overview," *Intelligent Information Technology Application Workshops, 2008. IITAW '08.*, pp. 953–957, 2008.
3. E.-S. A. Abuosba K.A, "Formalizing Service-Oriented Architectures," *IT Professional*, pp. 34–38, 2008.
4. P. Murray, "Spaghetti code." http://c2.com/cgi/wiki?SpaghettiCode.
5. S. P. Ahuja and A. Patel, "Enterprise service bus: A performance evaluation," *Communications and Network*, pp. 133–140, 2011.
6. "Github main page." https://github.com/.
7. J. Fenner, "Enterprise Application Integration Techniques." http://www.cs.ucl.ac.uk/staff/ucacwxe/lectures/3C05-02-03/aswe21-essay.pdf, 2003.
8. A. Mehta, "11 Of Best Opensource ESB Tools." http://www.toolsjournal.com/integrations-articles/item/224-11-of-best-opensource-esb-tools, 2011.
9. J. Wu and X. Tao, "Research of enterprise application integration based-on esb," *Advanced Computer Control (ICACC), 2010 2nd International Conference*, vol. 5, pp. 90–93, 27-29 March 2010.
10. A. Perera, "WSO2 ESB Performance Testing ." http://wso2.org/library/1721, 2007.
11. "Understanding Integration From A "Needs-Based" Perspective - Mule vs. JBoss ESB." http://www.mulesoft.org/comparing-mule-vs-jboss-esb, 2010.
12. "Understanding Integration From A "Needs-Based" Perspective - Mule vs. Open ESB / Glassfish ESB." http://www.mulesoft.org/comparing-mule-vs-open-esb-glassfish-esb, 2010.
13. "Understanding Integration From A "Needs-Based" Perspective - Mule vs. ServiceMix / Fuse ESB." http://www.mulesoft.org/comparing-mule-vs-servicemix-fuse-esb, 2010.
14. "Mule Performance Test Results." http://www.mulesoft.com/downloads/Whitepaper_perf_test_results.pdf, 2008.
15. "Jboss esb." http://www.jboss.org/jbossesb.
16. "Apache servicemix." http://servicemix.apache.org/home.html.
17. "Openesb." http://wiki.open-esb.java.net/.
18. "Mule esb." http://www.mulesoft.org/.
19. "Wso2 esb." http://wso2.com/products/enterprise-service-bus/.
20. "Fuse esb." http://fusesource.com/products/enterprise-servicemix/.
21. "Talend esb." http://www.talend.com/products/open-studio-esb.php.
22. "Petals esb." http://petals.ow2.org/download-petals-esb.html.
23. "Blackbird esb." https://www.ohloh.net/p/7246.
24. "Chainbuilder esb." http://sourceforge.net/projects/bostech-cbesb/.
25. "Spagic esb." http://www.spagoworld.org/xwiki/bin/view/Spagic/.
26. K. Vollmer, "The Forrester Wave: Enterprise Service Bus, Q2 2011." http://www.oracle.com/us/corporate/analystreports/infrastructure/forrester-wave-esb-q2-2011-395900.pdf, 2011.
27. A. Perera, "WSO2 ESB Performance Testing Round 2." http://wso2.org/library/2259, 2007.

28. A. Perera, "WSO2 Enterprise Service Bus (ESB) Performance Testing - Round 3." http://wso2.org/library/3740, 2008.
29. W. He and L. Xu, "Integration of distributed enterprise applications: A survey," *Industrial Informatics, IEEE Transactions*, vol. 99, pp. 1–1.
30. N. M. A. I. Alghamdi, A. and K. Nafjan, "An interoperability study of esb for c4i systems," *Information Technology (ITSim), 2010 International Symposium*, pp. 733 – 738, 15-17 June 2010.
31. P. Brebner, "Service-oriented performance modeling the mule enterprise service bus (esb) loan broker application," *Software Engineering and Advanced Applications, 2009. SEAA '09. 35th Euromicro Conference*, pp. 404 – 411, 27-29 Aug. 2009.
32. M.-C. M. Garcia-Jimenez F.J and G.-S. A.F, "Evaluating Open Source Enterprise Service Bus," *e-Business Engineering (ICEBE), 2010 IEEE 7th International Conference*, pp. 284–291, 10-12 Nov. 2010.
33. G. H. Nasr, K.A. and A. van Deursen, "Adopting and evaluating service oriented architecture in industry," *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference*, pp. 11–20, 15-18 March 2010.
34. "REST for the rest of us." http://developer.mindtouch.com/REST/REST_for_the_Rest_of_Us, 2008.
35. "The Grinder, a Java Load Testing Framework." http://grinder.sourceforge.net/.
36. "What is an ESB and why would you use one?." http://www.mulesoft.com/mule-esb-features.
37. "JSR 224: Java API for XML-Baswed WebServices (JAX-WS) 2.0." http://jcp.org/en/jsr/detail?id=224.